

ALIBABA CLOUD

Alibaba Cloud

Apsara Stack Enterprise

MaxCompute
User Guide

Product Version: v3.16.2

Document Version: 20220830

 Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).
5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
 Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 Danger: Resetting will result in the loss of user configuration data.
 Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
 Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	 Notice: If the weight is set to 0, the server no longer receives new requests.
 Note	A note indicates supplemental instructions, best practices, tips, and other content.	 Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click Settings > Network > Set network type .
Bold	Bold formatting is used for buttons, menus, page names, and other UI elements.	Click OK .
Courier font	Courier font is used for commands	Run the <code>cd /d C:/window</code> command to enter the Windows system folder.
<i>Italic</i>	Italic formatting is used for parameters and variables.	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] or [a b]	This format is used for an optional value, where only one item can be selected.	<code>ipconfig [-all -t]</code>
{ } or {a b}	This format is used for a required value, where only one item can be selected.	<code>switch {active stand}</code>

Table of Contents

- 1.What is MaxCompute ----- 37
- 2.Usage notes ----- 39
- 3.Preparations ----- 41
 - 3.1. Log on to the Apsara Uni-manager Management Console ----- 41
 - 3.2. Create an Apsara Stack tenant account ----- 41
 - 3.3. Create a project ----- 42
 - 3.4. Create a quota group ----- 43
- 4.Quick start ----- 45
 - 4.1. Overview ----- 45
 - 4.2. Configure the MaxCompute client ----- 46
 - 4.3. Add and delete a user ----- 47
 - 4.4. Grant and view permissions ----- 48
 - 4.4.1. Overview ----- 48
 - 4.4.2. ACL-based authorization ----- 48
 - 4.4.3. Policy-based access control ----- 49
 - 4.4.4. Query permissions ----- 50
 - 4.5. Create and authorize a role ----- 51
 - 4.6. Create or delete a table ----- 51
 - 4.6.1. Create a table ----- 51
 - 4.6.2. Query table information ----- 52
 - 4.6.3. Delete a table ----- 52
 - 4.7. Import or export data ----- 53
 - 4.8. Run SQL ----- 53
 - 4.8.1. Overview ----- 53
 - 4.8.2. SELECT ----- 53
 - 4.8.3. INSERT ----- 54

4.8.4. JOIN	54
4.8.5. Other limits	54
4.9. Compile and use UDFs	54
4.9.1. Overview	54
4.9.2. UDF example	55
4.9.3. UDAF example	55
4.9.4. UDTF example	56
4.10. Write and run a MapReduce program	57
4.11. Write and run a Graph job	58
4.12. Use Logview to view job running information	60
4.13. Use Logview V2.0 to view job running information	63
5. Basic concepts and common commands	71
5.1. Terms	71
5.2. Common commands	77
5.2.1. Overview	77
5.2.2. Project operations	77
5.2.3. Table operations	79
5.2.4. Instance operations	83
5.2.5. Resource operations	86
5.2.6. Function operations	88
5.2.7. Time zone configuration operations	90
5.2.8. Tunnel operations	91
5.2.9. Other operations	101
6. MaxCompute SQL	106
6.1. Overview	106
6.1.1. Scenarios	106
6.1.2. Reserved words	106
6.1.3. Partitioned table	106

6.1.4. Type conversion	107
6.1.4.1. Explicit conversions of data types	107
6.1.4.2. Implicit type conversion and its scope	108
6.1.4.3. SQL built-in functions	110
6.1.4.4. CASE WHEN	111
6.1.4.5. Conversions between the STRING and DATETIME ty...	111
6.2. Operators	111
6.2.1. Relational operators	112
6.2.2. Arithmetic operators	113
6.2.3. Bitwise operators	113
6.2.4. Logical operators	114
6.3. LOAD	115
6.4. DDL statements	117
6.4.1. Table operations	117
6.4.1.1. Create a table	117
6.4.1.2. Delete a table	123
6.4.1.3. Rename a table	123
6.4.1.4. Change the owner of a table	123
6.4.1.5. Modify the comment of a table	124
6.4.1.6. Modify the lifecycle settings of a table	124
6.4.1.7. Disable or enable the lifecycle feature	125
6.4.1.8. Modify the value of LastDataModifiedTime for a tab...	125
6.4.1.9. Modify the clustering attributes of a table	126
6.4.1.10. Delete data from a non-partitioned table	127
6.4.1.11. Archive table data	127
6.4.1.12. Forcibly delete data from a table or partition	128
6.4.2. View-based operation	128
6.4.2.1. Create a view	128

6.4.2.2. Delete a view	129
6.4.2.3. Rename a view	129
6.4.3. Column and partition operations	130
6.4.3.1. Add partitions	130
6.4.3.2. Delete a partition	130
6.4.3.3. Add columns or comments	133
6.4.3.4. Change the name of a column	133
6.4.3.5. Modify the comment of a column	134
6.4.3.6. Modify the name and comment of a column at the...	134
6.4.3.7. Change LastDataModifiedTime of a non-partitioned ...	135
6.4.3.8. Modify partition values	135
6.4.3.9. Merge partitions	136
6.5. DML statements	137
6.5.1. INSERT	137
6.5.1.1. Update data in a table	137
6.5.1.2. Insert data into multiple objects	140
6.5.1.3. Insert data into dynamic partitions	141
6.5.1.4. VALUES	143
6.5.2. SELECT	146
6.5.2.1. SELECT operations	146
6.5.2.2. Subquery	152
6.5.2.3. UNION ALL	152
6.5.2.4. JOIN	153
6.5.2.5. MAPJOIN hint	155
6.5.2.6. SELECT TRANSFORM	156
6.5.2.6.1. Overview	156
6.5.2.6.2. SELECT TRANSFORM examples	158
6.5.2.6.2.1. Call shell scripts	158

6.5.2.6.2.2. Call Python scripts	158
6.5.2.6.2.3. Call Java scripts	159
6.5.2.6.2.4. Call the scripts of other programming langu... ..	160
6.5.2.6.2.5. Call scripts in series	161
6.5.2.6.3. Performance	161
6.5.2.7. GROUPING SETS	162
6.5.2.7.1. Overview	162
6.5.2.7.2. Example	162
6.5.2.7.3. CUBE and ROLLUP	163
6.5.2.7.4. GROUPING and GROUPING_ID	164
6.5.2.8. UNION, INTERSECT, and EXCEPT	165
6.5.3. EXPLAIN	168
6.5.4. IF statement	171
6.5.5. UPDATE and DELETE	172
6.6. Built-in functions	179
6.6.1. Mathematical functions	180
6.6.1.1. ABS	180
6.6.1.2. ACOS	180
6.6.1.3. ASIN	181
6.6.1.4. ATAN	181
6.6.1.5. CEIL	181
6.6.1.6. CONV	182
6.6.1.7. COS	182
6.6.1.8. COSH	183
6.6.1.9. COT	183
6.6.1.10. EXP	183
6.6.1.11. FLOOR	184
6.6.1.12. LN	184

6.6.1.13. LOG	185
6.6.1.14. POW	185
6.6.1.15. RAND	186
6.6.1.16. ROUND	186
6.6.1.17. SIN	187
6.6.1.18. SINH	187
6.6.1.19. SQRT	187
6.6.1.20. TAN	188
6.6.1.21. TANH	188
6.6.1.22. TRUNC	188
6.6.1.23. Additional mathematical functions	189
6.6.1.23.1. Usage notes	189
6.6.1.23.2. LOG2	189
6.6.1.23.3. LOG10	190
6.6.1.23.4. BIN	190
6.6.1.23.5. HEX	191
6.6.1.23.6. UNHEX	191
6.6.1.23.7. RADIANS	191
6.6.1.23.8. DEGREES	192
6.6.1.23.9. SIGN	192
6.6.1.23.10. E	192
6.6.1.23.11. PI	193
6.6.1.23.12. FACTORIAL	193
6.6.1.23.13. CBRT	193
6.6.1.23.14. SHIFLEFT	194
6.6.1.23.15. SHIFTRIGHT	194
6.6.1.23.16. SHIFTRIGHTUNSIGNED	194
6.6.1.23.17. FORMAT_NUMBER	195

6.6.1.23.18. WIDTH_BUCKET	195
6.6.2. String functions	196
6.6.2.1. CHAR_MATCHCOUNT	196
6.6.2.2. CHR	197
6.6.2.3. CONCAT	197
6.6.2.4. INSTR	197
6.6.2.5. IS_ENCODING	198
6.6.2.6. KEYVALUE	199
6.6.2.7. LENGTH	200
6.6.2.8. LENGTHB	200
6.6.2.9. MD5	200
6.6.2.10. PARSE_URL	201
6.6.2.11. REGEXP_EXTRACT	201
6.6.2.12. REGEXP_INSTR	202
6.6.2.13. REGEXP_SUBSTR	203
6.6.2.14. REGEXP_COUNT	203
6.6.2.15. REGEXP_REPLACE	204
6.6.2.16. SPLIT_PART	204
6.6.2.17. SUBSTR	205
6.6.2.18. TOLOWER	206
6.6.2.19. TOUPPER	206
6.6.2.20. TO_CHAR	206
6.6.2.21. TRIM	207
6.6.2.22. LTRIM	207
6.6.2.23. RTRIM	208
6.6.2.24. REVERSE	208
6.6.2.25. SPACE	209
6.6.2.26. REPEAT	209

6.6.2.27. ASCII	209
6.6.2.28. Additional string functions	210
6.6.2.28.1. Usage notes	210
6.6.2.28.2. CONCAT_WS	210
6.6.2.28.3. LPAD	211
6.6.2.28.4. RPAD	211
6.6.2.28.5. REPLACE	212
6.6.2.28.6. SOUNDEX	212
6.6.2.28.7. SUBSTRING_INDEX	212
6.6.2.28.8. TRANSLATE	213
6.6.2.28.9. URL_ENCODE	213
6.6.2.28.10. URL_DECODE	214
6.6.2.28.11. JSON_TUPLE	214
6.6.2.28.12. FROM_JSON	216
6.6.2.28.13. TO_JSON	217
6.6.3. Date functions	218
6.6.3.1. DATEADD	218
6.6.3.2. DATEDIFF	219
6.6.3.3. DATEPART	220
6.6.3.4. DATETRUNC	220
6.6.3.5. GETDATE	221
6.6.3.6. ISDATE	221
6.6.3.7. LASTDAY	222
6.6.3.8. TO_DATE	222
6.6.3.9. TO_CHAR	223
6.6.3.10. UNIX_TIMESTAMP	223
6.6.3.11. FROM_UNIXTIME	224
6.6.3.12. WEEKDAY	224

6.6.3.13. WEEKOFYEAR	225
6.6.3.14. Additional date functions	225
6.6.3.14.1. Usage notes	225
6.6.3.14.2. YEAR	225
6.6.3.14.3. QUARTER	226
6.6.3.14.4. MONTH	226
6.6.3.14.5. DAY	227
6.6.3.14.6. DAYOFMONTH	227
6.6.3.14.7. HOUR	227
6.6.3.14.8. MINUTE	228
6.6.3.14.9. SECOND	228
6.6.3.14.10. FROM_UTC_TIMESTAMP	229
6.6.3.14.11. CURRENT_TIMESTAMP	229
6.6.3.14.12. ADD_MONTHS	229
6.6.3.14.13. LAST_DAY	230
6.6.3.14.14. NEXT_DAY	230
6.6.3.14.15. MONTHS_BETWEEN	231
6.6.3.14.16. EXTRACT	231
6.6.4. Window functions	232
6.6.4.1. Overview	232
6.6.4.2. COUNT	233
6.6.4.3. AVG	235
6.6.4.4. MAX	236
6.6.4.5. MIN	237
6.6.4.6. MEDIAN	237
6.6.4.7. STDDEV	237
6.6.4.8. STDDEV_SAMP	239
6.6.4.9. SUM	239

6.6.4.10. DENSE_RANK	241
6.6.4.11. RANK	242
6.6.4.12. LAG	243
6.6.4.13. LEAD	244
6.6.4.14. PERCENT_RANK	245
6.6.4.15. ROW_NUMBER	245
6.6.4.16. CLUSTER_SAMPLE	246
6.6.4.17. NTILE	248
6.6.4.18. NTH_VALUE	249
6.6.4.19. CUME_DIST	251
6.6.4.20. FIRST_VALUE	252
6.6.4.21. LAST_VALUE	253
6.6.5. Aggregate functions	254
6.6.5.1. Expressions of filter conditions in aggregate functio...	254
6.6.5.2. COUNT	255
6.6.5.3. COUNT_IF	256
6.6.5.4. AVG	256
6.6.5.5. MAX	257
6.6.5.6. MIN	258
6.6.5.7. MEDIAN	258
6.6.5.8. STDDEV	259
6.6.5.9. STDDEV_SAMP	260
6.6.5.10. SUM	261
6.6.5.11. WM_CONCAT	261
6.6.5.12. PERCENTILE	262
6.6.5.13. Additional aggregate functions	263
6.6.5.13.1. Usage notes	263
6.6.5.13.2. COLLECT_LIST	263

6.6.5.13.3. COLLECT_SET	263
6.6.5.13.4. VARIANCE or VAR_POP	264
6.6.5.13.5. VAR_SAMP	264
6.6.5.13.6. COVAR_POP	265
6.6.5.13.7. COVAR_SAMP	266
6.6.5.13.8. ANY_VALUE	266
6.6.5.13.9. NUMERIC_HISTOGRAM	267
6.6.5.13.10. PERCENTILE_APPROX	267
6.6.5.13.11. APPROX_DISTINCT	268
6.6.6. Other functions	268
6.6.6.1. ARRAY	268
6.6.6.2. ARRAY_CONTAINS	268
6.6.6.3. CAST	269
6.6.6.4. COALESCE	269
6.6.6.5. DECODE	269
6.6.6.6. EXPLODE	271
6.6.6.7. GET_IDCARD_AGE	271
6.6.6.8. GET_IDCARD_BIRTHDAY	272
6.6.6.9. GET_IDCARD_SEX	272
6.6.6.10. GREATEST	272
6.6.6.11. INDEX	273
6.6.6.12. MAX_PT	274
6.6.6.13. ORDINAL	274
6.6.6.14. LEAST	275
6.6.6.15. SIZE	275
6.6.6.16. SPLIT	276
6.6.6.17. STR_TO_MAP	276
6.6.6.18. UUID	277

6.6.6.19. UNIQUE_ID	277
6.6.6.20. SAMPLE	277
6.6.6.21. CASE WHEN expression	278
6.6.6.22. IF	279
6.6.6.23. Additional functions	279
6.6.6.23.1. Usage notes	279
6.6.6.23.2. MAP	280
6.6.6.23.3. MAP_KEYS	280
6.6.6.23.4. MAP_VALUES	281
6.6.6.23.5. SORT_ARRAY	281
6.6.6.23.6. POSEXPLODE	282
6.6.6.23.7. STRUCT	282
6.6.6.23.8. NAMED_STRUCT	283
6.6.6.23.9. INLINE	283
6.6.6.23.10. BETWEEN AND expression	284
6.6.6.23.11. NVL	285
6.6.6.23.12. TABLE_EXISTS	285
6.6.6.23.13. PARTITION_EXISTS	286
6.7. UDFs	286
6.7.1. Overview	286
6.7.2. Types of parameters and return values	287
6.7.3. UDF	290
6.7.4. UDAF	291
6.7.5. UDTFs	294
6.7.5.1. Overview	294
6.7.5.2. Examples of using UDTFs	295
6.7.6. Python UDFs	299
6.7.6.1. Before you start	299

6.7.6.2. Limits	299
6.7.6.3. Third-party libraries	300
6.7.6.4. Types of parameters and return values	300
6.7.6.5. UDF	302
6.7.6.6. UDAF	303
6.7.6.7. UDTF	303
6.7.6.8. Resource reference	304
6.7.7. Python 3 UDF	305
6.7.8. SQL Function	306
6.7.9. Embedded UDF	308
6.7.9.1. Background information	308
6.7.9.2. Feature summary	308
6.7.9.3. Limits	309
6.7.9.4. Examples	309
6.8. UDTs	311
6.8.1. Scenarios and limits	311
6.8.2. Feature summary	312
6.8.3. Implementation principles and feature description	312
6.8.4. Benefits	316
6.8.5. Performance advantages	316
6.8.6. Security advantages	316
6.8.7. More examples	316
6.8.7.1. Example of using Java arrays	316
6.8.7.2. Example of using JSON	317
6.8.7.3. Example of using complex data types	317
6.8.7.4. Aggregation example	318
6.8.7.5. Example of using table-valued functions	319
6.9. UDJ	319

6.9.1. Background information	319
6.9.2. Examples of using UDJ	320
6.9.2.1. Cross join operation by using UDJ	320
6.9.2.2. Pre-sorting	324
6.9.3. Performance	325
6.10. Parameterized view	326
6.11. CLONE TABLE	329
6.12. Geographic functions	331
6.12.1. Usage notes	331
6.12.2. Constructors	331
6.12.2.1. ST_AsBinary	331
6.12.2.2. ST_AsGeojson	332
6.12.2.3. ST_AsJson	332
6.12.2.4. ST_AsShape	332
6.12.2.5. ST_AsText	333
6.12.2.6. ST_GeomCollection	333
6.12.2.7. ST_GeomFromGeojson	333
6.12.2.8. ST_GeomFromJSON	334
6.12.2.9. ST_GeomFromShape	334
6.12.2.10. ST_GeomFromText	334
6.12.2.11. ST_GeomFromWKB	335
6.12.2.12. ST_GeometryType	335
6.12.2.13. ST_LineString	336
6.12.2.14. ST_LineFromWKB	336
6.12.2.15. ST_MultiLineString	336
6.12.2.16. ST_MLineFromWKB	336
6.12.2.17. ST_MultiPoint	337
6.12.2.18. ST_MPointFromWKB	337

6.12.2.19. ST_MultiPolygon	337
6.12.2.20. ST_MPolyFromWKB	338
6.12.2.21. ST_Point	338
6.12.2.22. ST_PointFromWKB	338
6.12.2.23. ST_PointZ	339
6.12.2.24. ST_Polygon	339
6.12.2.25. ST_PolyFromWKB	339
6.12.2.26. ST_SetSRID	340
6.12.3. Accessors	340
6.12.3.1. ST_Area	340
6.12.3.2. ST_Centroid	340
6.12.3.3. ST_CoordDim	341
6.12.3.4. ST_Dimension	341
6.12.3.5. ST_Distance	342
6.12.3.6. ST_GeodesicLengthWGS84	342
6.12.3.7. ST_GeometryN	343
6.12.3.8. ST_Is3D	343
6.12.3.9. ST_IsClosed	344
6.12.3.10. ST_IsEmpty	344
6.12.3.11. ST_IsMeasured	344
6.12.3.12. ST_IsSimple	345
6.12.3.13. ST_IsRing	345
6.12.3.14. ST_Length	345
6.12.3.15. ST_M	346
6.12.3.16. ST_MaxM	346
6.12.3.17. ST_MinM	346
6.12.3.18. ST_X	347
6.12.3.19. ST_Y	347

6.12.3.20. ST_Z	348
6.12.3.21. ST_MaxX	348
6.12.3.22. ST_MaxY	348
6.12.3.23. ST_MaxZ	349
6.12.3.24. ST_MinX	349
6.12.3.25. ST_MinY	350
6.12.3.26. ST_MinZ	350
6.12.3.27. ST_NumGeometries	351
6.12.3.28. ST_NumInteriorRing	351
6.12.3.29. ST_NumPoints	352
6.12.3.30. ST_PointN	352
6.12.3.31. ST_StartPoint	353
6.12.3.32. ST_EndPoint	353
6.12.3.33. ST_SRID	353
6.12.4. Operations	354
6.12.4.1. ST_Aggr_ConvexHull	354
6.12.4.2. ST_Aggr_Intersection	354
6.12.4.3. ST_Aggr_Union	354
6.12.4.4. ST_Bin	354
6.12.4.5. ST_BinEnvelope	355
6.12.4.6. ST_Boundary	355
6.12.4.7. ST_Buffer	355
6.12.4.8. ST_ConvexHull	356
6.12.4.9. ST_Difference	356
6.12.4.10. ST_Envelope	356
6.12.4.11. ST_ExteriorRing	357
6.12.4.12. ST_InteriorRingN	357
6.12.4.13. ST_Intersection	358

6.12.4.14. ST_SymmetricDiff	359
6.12.4.15. ST_Union	359
6.12.5. Relationship tests	359
6.12.5.1. ST_Contains	359
6.12.5.2. ST_Crosses	360
6.12.5.3. ST_Disjoint	360
6.12.5.4. ST_EnvIntersects	361
6.12.5.5. ST_Equals	361
6.12.5.6. ST_Intersects	361
6.12.5.7. ST_Overlaps	362
6.12.5.8. ST_Relate	362
6.12.5.9. ST_Touches	362
6.12.5.10. ST_Within	363
6.12.6. Geohash index functions	363
6.12.6.1. ST_GeoHash	363
6.12.6.2. ST_PointFromGeoHash	363
6.12.6.3. ST_EnvelopeFromGeoHash	364
6.12.6.4. ST_GeoHashNeighbours	364
6.12.7. S2 mesh functions	364
6.12.7.1. ST_S2CellIdsFromGeom	364
6.12.7.2. ST_S2CellIdsFromText	365
6.12.7.3. ST_S2CellCenterPoint	365
6.12.7.4. ST_S2CellNeighbours	365
6.12.8. Geodesic functions	365
6.12.8.1. ST_AreaWGS84	365
6.12.8.2. ST_DistanceWGS84	366
6.12.8.3. ST_BufferWGS84	366
6.12.8.4. ST_GeodesicDistance	366

6.12.8.5. ST_Distance_Sphere	367
6.12.8.6. ST_Area_Sphere	367
6.12.9. R-tree index functions	368
6.12.9.1. ST_BuildRtreeIndex	368
6.12.9.2. ST_ContainsFromRtree	368
6.12.9.3. ST_CrossesFromRtree	368
6.12.9.4. ST_EqualsFromRtree	369
6.12.9.5. ST_IntersectsFromRtree	369
6.12.9.6. ST_OverlapsFromRtree	369
6.12.9.7. ST_TouchesFromRtree	369
6.12.9.8. ST_WithinFromRtree	370
6.12.9.9. ST_KNNFromRtree	371
6.12.10. Other functions	372
6.12.10.1. ST_IsValid	372
6.12.10.2. ST_Transform	372
6.13. MaxCompute Hash Clustering	373
6.13.1. Background information	373
6.13.2. Descriptions	375
6.13.2.1. Enable Hash Clustering	375
6.13.2.2. Create a hash-clustered table	375
6.13.2.3. Modify table attributes	377
6.13.2.4. View and verify table attributes	377
6.13.3. Benefits	378
6.13.3.1. Bucket pruning and index optimization	378
6.13.3.2. Aggregation optimization	378
6.13.3.3. Storage optimization	378
6.13.4. ShuffleRemove	380
6.13.5. Limits	380

6.14. Common MaxCompute SQL parameter settings	381
6.14.1. Map configurations	381
6.14.2. Join configurations	381
6.14.3. Reduce configurations	381
6.14.4. UDF configurations	382
6.14.5. MapJoin configurations	382
6.14.6. Configurations of data skew	383
6.15. MapReduce-to-SQL conversion for execution	383
6.15.1. Overview	383
6.15.2. Configure the MaxCompute client	384
6.15.3. Configure running settings in DataWorks	384
6.15.4. View details	385
6.15.5. Perform operations on the distributed file system	386
6.16. Analysis of the mapping between SQL input and output ...	387
6.16.1. Overview	387
6.16.2. Usage notes	387
6.17. Common MaxCompute SQL errors and solutions	388
6.17.1. Data skew	388
6.17.1.1. Overview	388
6.17.1.2. GROUP BY data skew	389
6.17.1.3. DISTRIBUTE BY data skew	389
6.17.1.4. Data skew caused by JOIN operations	389
6.17.1.5. Data skew caused by multiple DISTINCT operations	390
6.17.1.6. Data skew caused by misuse of dynamic partitions	390
6.17.1.7. Use SKEWJOIN HINT to avoid skewed hot key value...	390
6.17.2. Insufficient computing resources	397
6.17.3. Methods to optimize storage in MaxCompute SQL	398
6.17.4. UDF OOM	399

6.18. Limits of MaxCompute SQL statements	400
6.19. Appendix	401
6.19.1. Escape characters	401
6.19.2. LIKE usage	402
6.19.3. Regular expressions	403
6.19.4. Reserved words and keywords	405
6.19.5. Settings of new data types	406
6.19.6. ACID semantics of MaxCompute parallel write jobs	406
7.MaxCompute Tunnel	410
7.1. Overview	410
7.2. Selection of tools to migrate data to the cloud	411
7.3. Introduction to the tools	411
7.4. Tunnel SDK overview	412
7.4.1. Overview	412
7.4.2. TableTunnel	413
7.4.3. InstanceTunnel	415
7.4.4. UploadSession	415
7.4.5. DownloadSession	417
7.4.6. TunnelBufferedWriter	418
7.5. Tunnel SDK example	419
7.5.1. Example of simple uploads	419
7.5.2. Example of simple downloads	420
7.5.3. Example of multi-thread uploads	422
7.5.4. Example of multi-thread downloads	424
7.5.5. Example of uploading data by using BufferedWriter	426
7.5.6. Example of uploading data by using BufferedWriter in ...	426
7.5.7. Examples of uploading and downloading data of comp...	427
7.6. Appendix	430

7.6.1. FAQ related to data upload and download by using Ma..	430
7.6.2. Common tunnel error codes	432
8. MaxCompute MapReduce	434
8.1. Overview	434
8.1.1. MapReduce	434
8.1.2. MapReduce 2	436
8.1.3. Compatibility with Hadoop MapReduce	436
8.2. Limits	441
8.3. Features	442
8.3.1. Run command	442
8.3.2. Concepts	443
8.3.2.1. Map/Reduce	443
8.3.2.2. Sorting	443
8.3.2.3. Partition	443
8.3.2.4. Combiner	444
8.3.3. Submit a job	444
8.3.4. Inputs and outputs	445
8.3.5. Use resources	446
8.3.6. Job running in local mode	446
8.4. SDK introduction	448
8.4.1. Overview of major MapReduce APIs	448
8.4.2. API description	449
8.4.2.1. MapperBase	449
8.4.2.2. ReducerBase	449
8.4.2.3. TaskContext	450
8.4.2.4. JobConf	451
8.4.2.5. JobClient	452
8.4.2.6. RunningJob	452

8.4.2.7. InputUtils	453
8.4.2.8. OutputUtils	453
8.4.2.9. Pipeline	453
8.4.3. Compatibility with Hadoop MapReduce	454
8.5. Data types	467
8.6. Sample programs	468
8.6.1. WordCount example	468
8.6.2. MapOnly example	469
8.6.3. MultipleInOut example	471
8.6.4. Multijobs example	474
8.6.5. SecondarySort example	476
8.6.6. Resource usage example	477
8.6.7. Counter usage example	479
8.6.8. Grep example	481
8.6.9. Join example	483
8.6.10. Sleep example	485
8.6.11. Unique example	486
8.6.12. Sort example	488
8.6.13. Examples of using partitioned tables as input	490
8.6.14. Pipeline example	491
9.MaxCompute Graph	494
9.1. Graph overview	494
9.1.1. Overview	494
9.1.2. Data structure of MaxCompute Graph	494
9.1.3. Graph logic	495
9.1.3.1. Graph loading	495
9.1.3.2. Iterative computing	496
9.1.3.3. Iteration termination	496

9.1.4. Aggregator mechanism	497
9.2. Graph feature overview	504
9.2.1. Run a job	504
9.2.2. Input and output	506
9.2.3. Read data from resources	507
9.2.3.1. Use GraphJob to specify resources to be read	507
9.2.3.2. Use resources in Graph	507
9.3. Graph SDK	507
9.4. Development and debugging	508
9.4.1. Development process	508
9.4.2. Development example	508
9.4.3. Perform debugging on the local computer	509
9.4.4. Temporary directory of a local job	511
9.4.5. Cluster debugging	512
9.4.6. Performance optimization	512
9.4.7. Built-in JAR packages	513
9.5. Limits	514
9.6. Sample programs	514
9.6.1. SSSP	514
9.6.2. PageRank	517
9.6.3. K-means clustering	519
9.6.4. BiPartiteMatchiing	523
9.6.5. Strongly connected components	526
9.6.6. Connected components	532
9.6.7. Topological sorting	535
9.6.8. Linear regression	537
9.6.9. Triangle count	541
9.6.10. Edge table import	543

9.6.11. Vertex table import	549
10. Java SDK	556
11. PyODPS	557
11.1. Getting started	557
11.2. Installation guide	558
11.3. Platform instructions	559
11.3.1. Overview	559
11.3.2. Use local PyODPS	559
11.3.3. Use PyODPS in DataWorks	560
11.4. Basic operations	562
11.4.1. Overview	562
11.4.2. Projects	562
11.4.3. Tables	562
11.4.4. SQL	568
11.4.5. Task instances	571
11.4.6. Resources	573
11.4.7. Functions	575
11.5. DataFrame	575
11.6. User experience enhancement	580
11.6.1. Command line	580
11.6.2. IPython	582
11.6.3. Jupyter Notebook	585
11.7. Configurations	587
11.8. API overview	590
11.9. FAQ	590
12. Java sandbox limits	593
13. Volume lifecycle management	594
13.1. Overview	594

13.2. Manage the lifecycle of a volume	594
14. Spark on MaxCompute	595
14.1. Overview	595
14.2. Project resources	595
14.3. Environment settings	595
14.3.1. Decompress the Spark on MaxCompute release packag...	595
14.3.2. Set environment variables	596
14.3.3. Configure Spark-defaults.conf	596
14.4. Quick start	597
14.5. Demo	599
14.6. Common cases	600
14.6.1. WordCount example	600
14.6.2. OSS access example	601
14.6.3. MaxCompute table read and write example	602
14.6.4. MaxCompute Table Spark-SQL example	603
14.6.5. Example of the self-developed client mode	605
14.6.6. MaxCompute Table PySpark example	605
14.6.7. MLlib example	606
14.6.8. PySpark interactive execution example	607
14.6.9. Spark-shell interactive execution example (read tables)	607
14.6.10. Spark-shell interactive execution example (Spark MLli...	607
14.6.11. Example of SparkR interactive execution	608
14.6.12. Example of PageRank with Apache Spark GraphX	609
14.6.13. Example of Spark Streaming-NetworkWordCount	610
14.7. Maven dependencies	611
14.8. Special notes	612
14.8.1. Running mode	612
14.8.2. Spark Streaming tasks	614

14.8.3. Job diagnosis	614
14.9. APIs supported by Spark	616
14.9.1. Spark Shell	616
14.9.2. Spark R	616
14.9.3. Spark SQL	617
14.9.4. Spark JDBC	617
14.10. Dynamic resource allocation of Spark	617
14.11. FAQ of Spark on MaxCompute	619
15. Elasticsearch on Maxcompute	621
15.1. Overview	621
15.2. Workflow	621
15.2.1. Overview	621
15.2.2. Distributed search workflow	621
15.2.3. Full-text index workflow	622
15.2.4. Authentication process	623
15.3. Quick start	623
15.4. Support for Elasticsearch applications	624
15.4.1. Typical practice of Elasticsearch on MaxCompute	624
15.4.2. Limits on Elasticsearch on MaxCompute in VPCs	625
15.5. Special notes	625
15.5.1. Find the Elasticsearch service domain name	625
16. Flink on MaxCompute	626
17. Non-structured data access and processing (integrated compu...	628
17.1. Overview	628
17.2. Internal data sources	628
17.2.1. OSS data source	628
17.2.1.1. Overview	628
17.2.1.2. Use a built-in extractor to read data from OSS	628

17.2.1.2.1. Overview	628
17.2.1.2.2. Create an external table	629
17.2.1.2.3. Query an external table	630
17.2.1.2.4. MSCK REPAIR TABLE	630
17.2.1.2.5. Read data from the CSV or TSV files compressed	631
17.2.1.3. Use a custom extractor to read data from OSS	633
17.2.1.3.1. Overview	633
17.2.1.3.2. Define a storage handler	633
17.2.1.3.3. Define an extractor	634
17.2.1.3.4. Compile and package code	635
17.2.1.3.5. Create an external table	636
17.2.1.3.6. Query an external table	636
17.2.1.4. Use a custom extractor to read external unstructured data	637
17.2.1.5. Data partitions	639
17.2.1.5.1. Overview	639
17.2.1.5.2. Standard organization method and directory structure	639
17.2.1.5.3. Custom directories of partition data in OSS	641
17.2.1.5.4. Access fully-customized non-partitioned data successfully	642
17.2.1.6. Output OSS data	642
17.2.1.6.1. Create an external table	642
17.2.1.6.2. Write data to a TSV text file by using an INSERT statement	642
17.2.1.6.3. Write data to an unstructured file by using an INSERT statement	644
17.2.1.6.4. Migrate data between different storage media by using an INSERT statement	644
17.2.1.7. STS mode authorization for OSS	645
17.2.2. Tablestore data source	646
17.2.2.1. Overview	646
17.2.2.2. Use MaxCompute to read and calculate data in Tablestore	647
17.2.2.2.1. Create an external table	647

17.2.2.2.2. Use an external table to access Tablestore data	648
17.2.2.3. Write data from MaxCompute to Tablestore	648
17.2.3. AnalyticDB data source	649
17.2.3.1. Overview	649
17.2.3.2. Write data to AnalyticDB	649
17.2.3.2.1. Create an external table	649
17.2.3.2.2. Write and query data	650
17.2.3.3. Read data from AnalyticDB for MySQL	650
17.2.4. RDS data source	651
17.2.4.1. Overview	651
17.2.4.2. Write data to RDS	651
17.2.4.2.1. Create an external table	651
17.2.4.2.2. Write and query data	652
17.2.4.3. Read data from ApsaraDB RDS	652
17.2.5. HDFS data source (Alibaba Cloud)	652
17.2.5.1. Overview	652
17.2.5.2. Data processing for common tables	653
17.2.5.2.1. Write data to HDFS	653
17.2.5.2.1.1. Create an external table	653
17.2.5.2.1.2. Write and query data	653
17.2.5.2.2. Read data from Apsara File Storage for HDFS	653
17.2.5.3. Data processing for partitioned tables	654
17.2.6. TDDL data source	655
17.2.6.1. Overview	655
17.2.6.2. Preparations	656
17.2.6.3. Create an external table	656
17.2.6.3.1. Syntax	656
17.2.6.3.2. Example	659

17.2.6.4. Read data from an external table	660
17.2.6.5. Write data to an external table	661
17.3. External data sources	661
17.3.1. HDFS data source (open-source)	661
17.3.1.1. Overview	661
17.3.1.2. Write data to HDFS	661
17.3.1.2.1. Create an external table	662
17.3.1.2.2. Write and query data	662
17.3.1.3. Read data from HDFS	662
17.3.2. MongoDB data source	663
17.3.2.1. Overview	663
17.3.2.2. Preparations	663
17.3.2.3. Write data to MongoDB	664
17.3.2.3.1. Create an external table	664
17.3.2.3.2. Write and query data	664
17.3.2.4. Read data from ApsaraDB for MongoDB	664
17.3.3. HBase data source	665
17.3.3.1. Overview	665
17.3.3.2. Write data to HBase	665
17.3.3.2.1. Write data to ApsaraDB for HBase by using an...	665
17.3.3.2.1.1. Create an external table	665
17.3.3.2.1.2. Write and query data	666
17.3.3.2.2. Write data to ApsaraDB for HBase by using bu...	666
17.3.3.3. Read data from ApsaraDB for HBase	669
18. Unstructured data access and processing (inside MaxCompute)	670
18.1. Overview	670
18.2. Create a volume external table	670
18.2.1. Syntax	670

18.2.2. Use a built-in storage handler to create a table	671
18.2.3. Use a custom StorageHandler to create a table	672
18.3. Access a Volume external table	672
19.Intelligent storage enhancement package	674
19.1. Backup and restoration	674
20.MaxCompute multi-region deployment	678
20.1. Overview	678
20.2. Features	678
20.3. Instructions	678
20.4. Examples of multi-region deployment	679
20.4.1. Synchronize table data among multiple clusters	679
20.4.2. Query the status of data synchronization between pri...	680
20.4.3. Cross-region direct read	682
20.4.4. Cross-region JOIN	684
21.Security solution	685
21.1. Intended users	685
21.2. Quick start	685
21.3. User authentication	688
21.4. Project user and authorization management	688
21.4.1. Overview	688
21.4.2. User management	688
21.4.3. Role management	689
21.4.4. ACL-based authorization actions	689
21.4.5. View permissions	692
21.5. Cross-project resource sharing	693
21.5.1. Overview	693
21.5.2. Package usage	693
21.5.2.1. Operations for package creators	694

21.5.2.2. Operations of package users	695
21.6. Project protection	696
21.6.1. Overview	696
21.6.2. Project data protection	696
21.6.3. Data export methods after project data protection is	696
21.6.4. Package-based resource sharing and project data prot... ..	698
21.7. Project security configurations	699
21.8. Authorization policies	699
21.8.1. Policy overview	699
21.8.2. Policy-related terminology	701
21.8.3. Access policy structure	702
21.8.3.1. Overview	702
21.8.3.2. Authorization statement structure	703
21.8.3.3. Structure of condition blocks	703
21.8.3.4. Condition action types	703
21.8.3.5. Condition keys	704
21.8.4. Access policy norm	705
21.8.4.1. Principal naming conventions	705
21.8.4.2. Naming conventions of resources	705
21.8.4.3. Naming conventions of actions	706
21.8.4.4. Naming conventions of condition keys	707
21.8.4.5. Example of a policy for policy-based authorization	707
21.8.5. Differences between policy-based access control and A... ..	707
21.8.6. Limits	708
21.9. Collection of security statements	709
21.9.1. Project security configurations	709
21.9.2. Project permission management	710
21.9.3. Package-based resource sharing	711

22.Hierarchical throttling	713
22.1. Overview	713
22.2. Hierarchical throttling	713
22.3. Specify the maximum number of instances that are allow.....	713
23.Frequently-used tools	716
23.1. MaxCompute console	716
23.1.1. Usage notes	716
23.1.2. Install the MaxCompute client	716
23.1.3. Configuration-related operations	716
23.2. Eclipse development plugin	720
23.2.1. Install the Eclipse development plug-in	720
23.2.2. Create a project	722
23.2.2.1. Method 1	722
23.2.2.2. Method 2	725
23.2.3. MapReduce running example	726
23.2.3.1. Run a WordCount program	726
23.2.3.2. Run a custom MapReduce program	729
23.2.4. UDF development and running example	741
23.2.4.1. Local debug UDF programs	741
23.2.4.1.1. Run a UDF from the menu bar	741
23.2.4.1.2. Use the right-click shortcut menu to run a UDF	742
23.2.4.2. Run a UDF	744
23.2.5. Graph operation example	746
24.MaxCompute feature enhancement packages	750
24.1. Content moderation	750
24.2. MCQA	751
24.2.1. Overview	751
24.2.2. Usage notes	753

24.3. VVP On MaxCompute	760
24.3.1. Overview	760
24.3.2. Benefits	760
24.3.3. Activate VVP On MaxCompute	761
24.3.4. Usage notes	762
24.4. Mars	768
24.4.1. Overview	768
24.4.2. Preparations	771
24.4.3. Usage notes	772
25. MaxCompute FAQ	775
26. Open source features of MaxCompute	781

1. What is MaxCompute

MaxCompute is a data processing platform developed by Alibaba Group to process large amounts of data. MaxCompute provides channels for data uploads and downloads, a range of computing and analysis features including MaxCompute SQL and MaxCompute MapReduce, and comprehensive security solutions.

MaxCompute is used to store and compute large amounts of structured data. It provides warehousing solutions for large amounts of data, as well as big data analytics and modeling services.

As data collection techniques are becoming increasingly diverse and comprehensive, industries are amassing larger amounts of data. The data amount has increased from 100 GB, 1 TB to even 1 PB, far exceeding the processing capabilities of traditional software. Analysis tasks for large amounts of data require distributed computing instead of reliance on a single server. However, distributed computing models require skilled data analysts and are difficult to maintain. To use a distributed computing model, data analysts must understand business requirements and underlying computing models.

MaxCompute aims to provide easy analysis and processing of large amounts of data. You can analyze big data without a deep knowledge of distributed computing. MaxCompute is widely implemented within Alibaba Group for multiple scenarios. The scenarios include data warehousing and BI analysis for large Internet enterprises, website log analysis, e-commerce transaction analysis, and exploration of user characteristics and interests.

MaxCompute provides the following features:

- Data channels
 - Tunnel: provides highly concurrent uploads and downloads of offline data. MaxCompute Tunnel enables you to upload or download large amounts of data to or from MaxCompute. MaxCompute Tunnel provides a Java API.
 - DataHub: provides real-time data uploads and downloads. Data uploaded by using DataHub is available immediately, while data uploaded by using MaxCompute Tunnel is not.
- Computing and analysis
 - SQL: MaxCompute stores data in tables and provides SQL query capabilities. MaxCompute can be used as traditional database software, but it is capable of processing terabytes and petabytes of data. MaxCompute SQL does not support transactions, indexes, or operations such as UPDATE and DELETE. The SQL syntax used in MaxCompute is different from that in Oracle and MySQL. SQL statements from other database engines cannot be seamlessly migrated to MaxCompute. MaxCompute SQL responds to queries within a few minutes or seconds, instead of milliseconds. MaxCompute SQL is easy to learn. You can get started with MaxCompute SQL based on your prior experience of database operations, without a deep knowledge of distributed computing.
 - MapReduce: First proposed by Google, MapReduce is a distributed data processing model that has gained extensive attention and been used in a wide range of business scenarios. This document briefly describes the MapReduce model. You must have a basic knowledge of distributed computing and relevant programming experience before you use MapReduce. MapReduce provides a Java API.
 - Graph: an iterative graph computing framework provided in MaxCompute. Graph computing jobs use graphs to build models. A graph is a collection of vertices and edges that have values. MaxCompute Graph iteratively edits and evolves graphs to obtain analysis results.
 - Unstructured data access and processing in integrated computing scenarios: MaxCompute SQL cannot directly process external data, such as unstructured data from Object Storage Service (OSS). Data must be imported to MaxCompute tables by using relevant tools before computation. The MaxCompute team introduces the unstructured data processing framework to the MaxCompute system architecture to handle this issue.

MaxCompute allows you to create external tables to process data from the following data sources:

- Internal data sources: OSS, Tablestore, AnalyticDB, ApsaraDB RDS, Alibaba Cloud HDFS, and TDDL
- External data sources: open source HDFS, MongoDB, and HBase

- Unstructured data access and processing in MaxCompute: MaxCompute reads and writes volumes to store and process unstructured data, which otherwise must be stored in an external storage system.
- Spark on MaxCompute: a big data analytics engine developed by Alibaba Cloud to provide big data processing capabilities for Alibaba Group, government agencies, and enterprises. For more information, see [Spark on MaxCompute](#).
- Elasticsearch on MaxCompute: an enterprise-class full-text retrieval system developed by Alibaba Cloud to retrieve large amounts of data. It provides near-real-time (NRT) search for government agencies and enterprises. For more information, see [Elasticsearch on MaxCompute](#).
- SDK: a toolkit provided for developers.
- Security solution: MaxCompute provides powerful security features to ensure data security.

2. Usage notes

You can selectively read topics in this document based on your requirements. This topic provides reading recommendations based on your roles.

MaxCompute beginners

If you are a beginner in MaxCompute, we recommend that you first familiarize yourself with the following topics:

- **What is MaxCompute:** This topic provides the MaxCompute overview and describes its features. It helps you obtain a general knowledge of MaxCompute.
- **Quick start:** This topic describes how to download and configure the client, create a table, grant permissions, import data, and export data. It also describes how to run SQL jobs, user-defined functions (UDFs), and MapReduce programs.
- **Terms and common statements:** This topic introduces the basic terms of MaxCompute and commonly used statements in MaxCompute. It helps you familiarize yourself with operations on MaxCompute.
- **Frequently used tools:** This topic describes how to download, configure, and use the commonly used tools in MaxCompute before you analyze data.

Data analysts

If you are a data analyst, we recommend that you familiarize yourself with the following topics:

MaxCompute SQL: The topic describes how to query and analyze large amounts of data stored in MaxCompute.

MaxCompute SQL provides the following features:

- Allows you to use the CREATE, DROP, and ALTER DDL statements to manage tables and partitions.
- Allows you to execute the following DML statements:
 - You can use the SELECT statement to select data records in a table and the WHERE clause to query data records that meet specific conditions. These statements help you search for data records.
 - You can join two tables by using equi-joins.
 - You can use the GROUP BY clause to aggregate columns.
 - You can use the INSERT OVERWRITE or INSERT INTO statement to insert data records into another table.
 - You can use SELECT TRANSFORM to simplify the reference of script code.
- Allows you to use built-in functions and UDFs to complete a variety of computations.
- Allows you to use user-defined types (UDTs) to reference classes or objects of third-party programming languages in SQL statements and therefore to obtain data or call methods.
- Allows you to use UJEs to implement flexible cross-table and multi-table custom operations and reduce the detailed operations on the underlying distributed systems by using methods such as MapReduce.
- Allows you to collect statistics on tables and configure table lifecycles.
- Supports regular expressions.

Users with development experience

If you have development experience, understand the distributed architecture, and want to obtain data analytics capabilities that SQL cannot deliver, we recommend that you read the following advanced functional topics of MaxCompute:

- **MaxCompute MapReduce:** a MapReduce programming model for Java. You can use the Java API provided by MapReduce to write MapReduce programs and process MaxCompute data.
- **MaxCompute Graph:** a processing framework for iterative graph computing. A graph consists of vertices and edges, both of which contain values. MaxCompute Graph iteratively edits and evolves graphs to obtain analysis results.

- Eclipse plug-in: an integrated development environment (IDE) to help you complete development by using MapReduce, UDFs, and Graph SDK for Java.
- SDK for Java: a toolkit provided for developers.
- MaxCompute Tunnel: enables you to upload or download large amounts of data to or from MaxCompute at a time.

Project owners or administrators

If you are a project owner or an administrator, we recommend that you familiarize yourself with the following topics:

Security solution: This topic describes how to authorize users, enable cross-project resource sharing, configure project protection, and perform policy-based authorization.

3. Preparations

3.1. Log on to the Apsara Uni-manager Management Console

This topic describes how to log on to the Apsara Uni-manager Management Console.

Prerequisites

- The URL of the Apsara Uni-manager Management Console is obtained from the deployment personnel before you log on to the Apsara Uni-manager Management Console.
- We recommend that you use the Google Chrome browser.

Procedure

1. In the address bar, enter the URL of the Apsara Uni-manager Management Console. Press the Enter key.
2. Enter your username and password.

Obtain the username and password that you can use to log on to the console from the operations administrator.

 **Note** When you log on to the Apsara Uni-manager Management Console for the first time, you must change the password of your username. Your password must meet complexity requirements. The password must be 10 to 32 characters in length and must contain at least two of the following character types:

- Uppercase or lowercase letters
- Digits
- Special characters, which include ! @ # \$ %

3. Click **Log On**.
4. If your account has multi-factor authentication (MFA) enabled, perform corresponding operations in the following scenarios:
 - It is the first time that you log on to the console after MFA is forcibly enabled by the administrator.
 - a. On the Bind Virtual MFA Device page, bind an MFA device.
 - b. Enter the account and password again as in Step 2 and click **Log On**.
 - c. Enter a six-digit MFA verification code and click **Authenticate**.
 - You have enabled MFA and bound an MFA device.

Enter a six-digit MFA authentication code and click **Authenticate**.

 **Note** For more information, see the *Bind a virtual MFA device to enable MFA* topic in *Apsara Uni-manager Operations Console User Guide*.

3.2. Create an Apsara Stack tenant account

This topic describes how to create an Apsara Stack tenant account.

Procedure

1. Log on to the Apsara Uni-manager Management Console as an administrator.
2. In the top navigation bar, choose **Products > Big Data > MaxCompute** to go to the **MaxCompute** homepage.
3. In the left-side navigation pane of the page that appears, click **Task Accounts**. In the upper-right corner of the page that appears, click **Create Task Account**.
4. In the dialog box that appears, configure the parameters and click **OK**.

Parameter	Description
Name	The name of the Apsara Stack tenant account.
Organization	The organization to which the Apsara Stack tenant account belongs.
Description	The description of the Apsara Stack tenant account. You can leave this parameter empty.

3.3. Create a project

This topic describes how to create a MaxCompute project.

Procedure

1. Log on to the Apsara Uni-manager Management Console as an administrator.
2. In the top navigation bar, choose **Products > Big Data > MaxCompute** to go to the **MaxCompute** homepage.
3. In the left-side navigation pane, click **Projects**. In the upper-right corner of the page that appears, click **Create MaxCompute Project**.
4. On the page that appears, configure the parameters and click **Submit**.

Configurations in the Region section

Parameter	Description
Organization	The organization that you select for the project.
Resource Set	The resource set of the selected organization.
Cross-domain Enhancement Pack	The region with which the project you create can associate.
VPC	The virtual private cloud (VPC) in the region. The VPC is used to isolate networks.
Cluster	The information about the cluster of the project.

Configurations in the Security Enhancement Switch section

Parameter	Description
Enhancement Pack for Joint Computing	By default, this switch is turned off. You can turn on the switch to support joint computing features.
Encrypted	Specifies whether to encrypt data. Default value: No.

Parameter	Description
Encryption Method	By default, this parameter is not displayed. It is displayed when you set Encrypted to Yes. Valid values: AESCTR, AES256, RC4, and SM4.
Encryption Key	By default, this parameter is not displayed. It is displayed when you set Encrypted to Yes. Default value: Generate Automatically.

Configurations in the Resource Configurations section

Parameter	Description
Quota Group	The quota group that corresponds to the cluster. If no groups are available, click Create Quota Group to create one. For more information about the configurations, see Create a quota group .
Storage	You need only to specify the Requested (GB) parameter. The Requested (GB) parameter specifies the storage space that is requested for the project. The value cannot be greater than the total capacity of the disk.

Configurations in the Basic Settings section

Parameter	Description
Project Name	The name of the project you want to create.
Owner Account	The name of the Apsara Stack tenant account that corresponds to the project owner. The value of this parameter depends on the associated organization. You cannot specify this parameter.
Task Account	Optional. The account of the task. If no accounts are available, click Create Task Account to create one. For more information about the creation process and detailed configurations, see Prepare an Apsara Stack tenant account .
Rule	The description of the project. This parameter can be empty.

3.4. Create a quota group

This topic describes how to create a quota group in MaxCompute.

Procedure

1. Log on to the Apsara Uni-manager Management Console as an administrator.
2. In the top navigation bar, choose **Products > Big Data > MaxCompute** to go to the **MaxCompute** homepage.
3. In the left-side navigation pane, click **Quota Groups**. In the upper-right corner of the page that appears, click **Create**.

4. On the page that appears, configure the parameters and click **Submit**.

Configurations in the Region section

Parameter	Description
Organization	The organization that you select for the quota group.
Resource Set	The resource set in the selected organization.
Region	The region of the cluster where the quota group resides.
Cluster	The information about the cluster of the quota group.

Configurations in the Basic Settings section

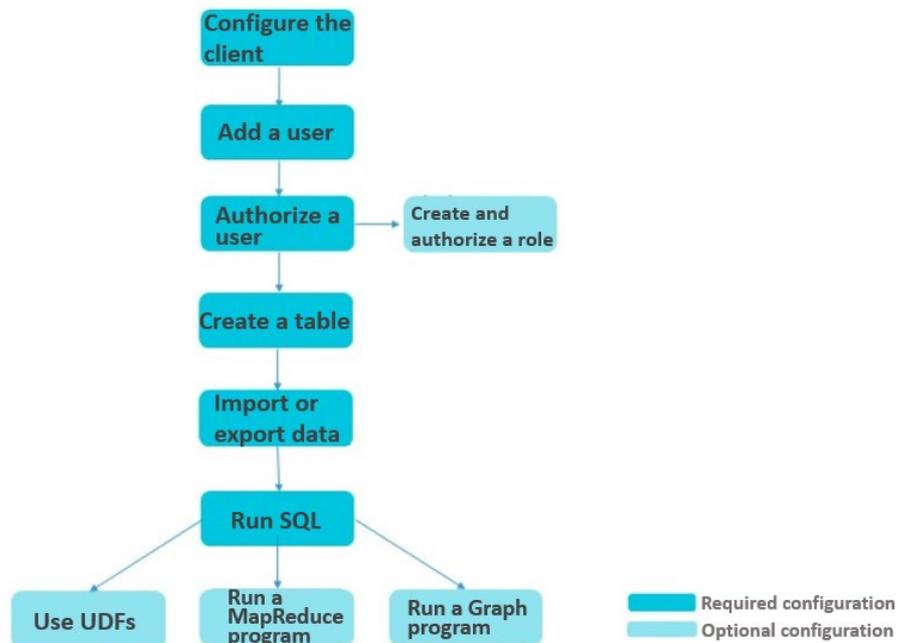
Parameter	Description
Quota Group Name	The name of the quota group that you want to create.
Sharing Scope	The sharing scope of the quota group. Valid values: Current Resource Set, Current Organization and Subordinate Organizations, and Current Organization.
Quantity	The number of resources to allocate to the current quota group. The value must be greater than 0.

4. Quick start

4.1. Overview

This topic describes the procedure to use MaxCompute. It provides you with step-by-step instructions on basic MaxCompute operations.

The following figure shows the procedure to use MaxCompute.



1. Configure the client.

You must install and configure the client to access MaxCompute projects and use all the features of MaxCompute. For more information, see [MaxCompute client](#).

2. Add a user.

Except for the project owner, all users must be manually added to a project and granted permissions before they can perform operations on the project.

3. Authorize a user.

After you add a user, you must authorize the user to perform operations on the project. A user can perform operations on the project only after the user is authorized.

4. (Optional) Create and authorize a role.

It is time-consuming to authorize each user in a project that contains a large number of users. Project administrators can use roles to grant users a specified set of permissions. After you authorize a role, all users who assume this role are granted the same permissions.

5. Create a table.

After a user is added to a project and authorized, the user can use MaxCompute. Table operations are the most basic operations in MaxCompute.

6. Import or export data.

You can use the SDK provided by MaxCompute Tunnel to write your own Java programs and import or export data.

7. Execute SQL statements.

You can familiarize yourself with the limits on a few common SQL statements. For more information about how to execute SQL statements, see [MaxCompute SQL](#).

8. Perform the following optional operations:

o Use UDFs.

After you install the MaxCompute client, you can try to use user-defined functions (UDFs). MaxCompute supports user-defined scalar functions, user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs).

o Run a MapReduce program.

After you install the MaxCompute client, you can run a MapReduce program.

o Run a Graph program.

After you install the MaxCompute client, you can run a Graph program.

4.2. Configure the MaxCompute client

The MaxCompute client allows you to access MaxCompute projects and use MaxCompute features. This topic describes how to download, configure, and run the MaxCompute client.

Prerequisites

JRE 1.8 is installed on your on-premises machine because the MaxCompute client is developed in Java. In addition, you have an Apsara Stack tenant account and have obtained the AccessKey ID and AccessKey secret.

 **Note** Before you configure the MaxCompute client, make sure that a project is created and the AccessKey ID and AccessKey secret are obtained.

Procedure

1. Download the [client](#) package to your on-premises machine.
2. Decompress the package to the folder in which you want to store the client. The package contains the following folders:

```
bin/
conf/
lib/
plugins/
```

3. Edit the `odps_config.ini` file in the `conf` folder. The following code shows the configuration information:

```
project_name=my_project
access_id=*****
access_key=*****
end_point= <Endpoint of MaxCompute>
```

- o `access_id` and `access_key` are the AccessKey pair of the Apsara Stack tenant account. `access_id` corresponds to the AccessKey ID and `access_key` corresponds to the AccessKey secret.
 - o `project_name=my_project` specifies the project that you want to access. This is the default project that is accessed each time you log on to the client. If this parameter is not configured, you must run the `use project_name` command to access the project after you log on to the client.
 - o Set `end_point` to the endpoint of MaxCompute. The endpoint varies based on the user account.
4. After the modification, run the `odpscmd` file in Linux or the `odpscmd.bat` file in Windows in the `bin` directory to execute SQL statements and related commands. Example:

```
create table tbl1(id bigint);
insert overwrite table tbl1 select count(*) from tbl1;
select 'welcome to MaxCompute!' from tbl1;
```

 **Note** For more information about SQL statements, see [MaxCompute SQL](#).

4.3. Add and delete a user

Except for a project owner, all users must be manually added to the project and granted permissions before they can perform operations on the project. This topic describes how to add users to or delete users from a project as a project owner.

If you are a project owner, we recommend that you read this topic in full. If you are a regular user, we recommend that you submit an application to a project owner to join a project, and read the subsequent topics after you are added to the project.

Add a user

Syntax:

```
add user <full_username>;
```

Examples:

```
add user bob@aliyun.com;
```

If you are not sure whether the user is already in the project, run the following command to query the users in the project:

```
list users;
```

Note

- After a user is added to a MaxCompute project, the user must be authorized by the project owner. Then, the user can perform authorized operations on the project.
- For more information about authorization, see [Grant and view permissions](#).

Delete a user

Syntax:

```
remove user <full_username> ;
```

Examples:

```
remove user bob@aliyun.com;
```

 Note

- Before you delete a user, make sure that you have revoked all the permissions from the user. If you delete a user before you revoke the permissions from the user, the permissions are retained. If the user is added to the project again, the user still has the permissions that were previously granted.
- For more information about how to add or delete users, see [User management](#).

4.4. Grant and view permissions

4.4.1. Overview

After you add a user, you must authorize the user to perform operations on the project. A user can perform operations on the project only after the user is authorized.

Authorization is a process of granting permissions to perform operations on objects, such as tables, tasks, and resources, in MaxCompute. The permissions include read, write, and view permissions.

The following topics are intended for project administrators. If you are a regular MaxCompute user, make sure that you have obtained the required permissions.

MaxCompute provides two authorization mechanisms. For more information, see [ACL authorization](#) and [Policy authorization](#).

4.4.2. ACL-based authorization

This topic describes the statements for ACL-based authorization and provides examples.

ACL-based authorization in MaxCompute applies to the following objects: projects, tables, functions, resources, instances, and tasks. Each type of object requires different operation permissions. For more information, see [ACL-based authorization actions](#).

Syntax:

```
grant actions on object to subject
revoke actions on object from subject
actions ::= action_item1, action_item2, ...
object ::= project project_name | table schema_name |
          instance inst_name | function func_name |
          resource res_name
subject ::= user full_username | role role_name
```

Examples:

alice@aliyun.com is newly added to the test_project_a project. Allen is a RAM user of bob@aliyun.com. An Alibaba Cloud account can execute the following statements to grant permissions, including the permissions to submit jobs, create data tables, and query existing objects in a project:

```
-- Enter a project.
use test_project_a;
-- Add a member to the project.
add user aliyun$alice@aliyun.com;
-- Add a RAM user.
add user ram$bob@aliyun.com:Allen;
-- Create a role named worker.
create role worker;
-- Assign the worker role to the added members.
grant worker TO aliyun$alice@aliyun.com;
grant worker TO ram$bob@aliyun.com:Allen;
-- Grant the CREATE INSTANCE, CREATE RESOURCE, CREATE FUNCTION, CREATE TABLE, and LIST permissions to
the worker role.
grant CreateInstance, CreateResource, CreateFunction, CreateTable, List ON PROJECT test_project_a TO
ROLE worker;
-- Grant all instance permissions to the worker role.
grant all on instance instance_name to Role worker;
```

4.4.3. Policy-based access control

This topic describes the commands that you can use for policy-based access control and provides examples on how to configure a policy.

Policy-based access control is subject-based authorization. For more information, see [Authorization policies](#).

Commands:

```
GET POLICY;
-- Read the project policy.
PUT POLICY <policyFile>;
-- Specify or overwrite the project policy.
GET POLICY ON ROLE <roleName>;
-- Read the policy of a role in the project.
PUT POLICY <policyFile> ON ROLE <roleName>;
-- Specify or overwrite the policy of a role in the project.
```

Example on how to configure a policy:

```

{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "odps:*"
      ],
      "Resource": "acs:odps*:projects/$user_project_name/tables/*",
      "Condition": {
        "StringEquals": {
          "odps:TaskType": [
            "DT"
          ]
        }
      },
    },
    {
      "Effect": "Allow",
      "Action": [
        "odps:List",
        "odps:Read",
        "odps:Describe",
        "odps:Select"
      ],
      "Resource": [
        "acs:odps*:projects/$user_project_name/tables/a",
        "acs:odps*:projects/$user_project_name"
      ],
      "Condition": {
        "StringEquals": {
          "odps:TaskType": [
            "SQL"
          ]
        }
      }
    }
  ]
}

```

 **Note** The preceding example disables the Tunnel feature of \$user_project_name, and grants the user the LIST, READ, DESCRIBE, and SELECT permissions only on the project and table a in the project.

4.4.4. Query permissions

You can execute a statement to query user permissions in MaxCompute.

Execute the following statement to query the permissions of a user:

```
show grants for $user_name;
```

 **Note** For more information, see [View permissions](#).

4.5. Create and authorize a role

If a project has a large number of users, the authorization process is time-consuming. Project administrators can use roles to grant users a specified set of permissions. After you authorize a role, all users who assume this role are granted the same permissions. This topic describes how to create and authorize a role.

One user can assume multiple roles, and multiple users can assume the same role.

Create a role

Syntax:

```
create role <roleName>;
```

Examples:

```
create role player;
```

Assign a role to a user

Syntax:

```
grant <roleName> to <full_username>;
```

Examples:

```
grant player to bob@aliyun.com;
```

Delete a role

Syntax:

```
drop role <roleName>;
```

Examples:

```
drop role player;
```

 **Note** Before you delete a role, make sure that all users have been removed from the role.

Authorize a role

Role authorization is similar to user authorization. For more information about how to authorize roles, see [User authorization](#). For more information about role authorization, see [Role management](#).

4.6. Create or delete a table

4.6.1. Create a table

This topic describes how to create a table.

Syntax:

```
create table [if not exists] table_name
[(col_name data_type [comment col_comment], ...)] [comment table_comment]
[partitioned by (col_name data_type [comment col_comment], ...)] [lifecycle days]
[as select_statement]
-- You can also execute the following statement to create a table:
create table [if not exists] table_name
like existing_table_name
```

Examples:

```
create table test1 (key string);
-- Create a non-partitioned table.
create table test2 (key bigint) partitioned by (pt string, ds string);
-- Create a partitioned table.
create table test3 (key boolean) partitioned by (pt string, ds string) lifecycle 100;
-- Create a table that has a lifecycle specified.
create table test4 like test3;
-- Create a table named test4 that has the same attributes, such as the field type and partition type,
as those of test3. The only difference between the two tables is the lifecycle.
create table test5 as select * from test2;
-- Create a table named test5. However, the partitioning and lifecycle attributes are not replicated
to the destination table. Only data of test2 is replicated to test5.
```

You can specify partitions and lifecycles for MaxCompute tables. For more information about how to create a table, see [Create a table](#). For more information about how to modify partitions, see [Add a partition](#) and [Delete a partition](#). For more information about how to modify lifecycle settings, see [Modify the lifecycle of a table](#).

4.6.2. Query table information

This topic describes how to query table information.

Syntax:

```
desc <table_name>;
```

Examples:

```
desc test3;
-- Query information about test3.
desc test4;
-- Query information about test4.
```

4.6.3. Delete a table

This topic describes how to delete a table.

Syntax:

```
drop table [if exists] table_name;
```

Examples:

```
drop table test2;
```

 **Note** For more information, see [Delete a table](#).

4.7. Import or export data

This topic describes how to compile Java programs by using the MaxCompute Tunnel SDK and import data to or export data from MaxCompute.

For more information about data import and export, see [Tunnel SDK example](#).

4.8. Run SQL

4.8.1. Overview

This topic describes the limits on the execution of SQL statements.

For more information about how to execute SQL statements, see [MaxCompute SQL](#).

Take note of the following points when you use MaxCompute SQL:

- MaxCompute SQL does not support transactions, indexes, or operations such as UPDATE and DELETE.
- The SQL syntax of MaxCompute is different from that of Oracle or MySQL. You cannot directly migrate SQL statements from other database engines to MaxCompute.
- MaxCompute SQL does not respond to queries in real time. It requires a few minutes to return query results, instead of seconds or milliseconds.

4.8.2. SELECT

This topic describes limits of the SELECT statements.

The following limits apply to the SELECT statements:

- The key of the GROUP BY clause can be the name of a column in the input table, or the expression composed of columns in the input table. However, it cannot be the output column from the SELECT operation.

```
select substr(col2, 2) from tbl group by substr(col2, 2);
-- This statement can be executed because the key of the GROUP BY clause is the expression composed of columns in the input table.
select col2 from tbl group by substr(col2, 2);
-- This statement cannot be executed because the key of the GROUP BY clause is not included in the columns of the SELECT operation.
select substr(col2, 2) as c from tbl group by c;
-- This statement cannot be executed because the key of the GROUP BY clause is the alias of an output column from the SELECT operation.
```

 **Note** In most cases, the GROUP BY clause precedes the SELECT operation during the parsing of SQL statements. Therefore, GROUP BY uses only the column names in the input table or expressions of columns as keys.

- The DISTRIBUTE BY clause must be placed before the SORT BY clause.
- The key of the ORDER BY, SORT BY, or DISTRIBUTE BY clause must be the alias of an output column from the SELECT operation.

```
select col2 as c from tbl order by col2 limit 100;
-- This statement cannot be executed because the key of the ORDER BY clause is not the alias of an
output column from the SELECT operation.
select col2 from tbl order by col2 limit 100;
-- This statement can be executed. If an output column from the SELECT operation does not have an
alias, the column name is used as the alias.
```

Note In most cases, the SELECT operation is performed before the ORDER BY, SORT BY, and DISTRIBUTE BY clauses during the parsing of SQL statements. Therefore, only the aliases of output columns from the SELECT operation can be used as keys of these clauses.

For more information about the SELECT statements, see [SELECT](#).

4.8.3. INSERT

This topic describes the limits of the INSERT statements.

The following limits apply to the INSERT statements:

- If you execute an INSERT statement to insert data into a partition, the partition key columns cannot be included in the SELECT clause.

```
insert overwrite table sale_detail_insert partition (sale_date='2017', region='china') select shop
_name, customer_id, total_price, sale_date, region from sale_detail;
-- An error is returned. The sale_date and region columns are partition key columns and cannot be
included in the INSERT statement that is used to insert data into static partitions.
```

- If you execute an INSERT statement to insert data into dynamic partitions, the dynamic partition key columns must be included in the SELECT clause.

```
insert overwrite table sale_detail_dypart partition (sale_date='2017', region) select shop_name,cu
stomer_id,total_price from sale_detail;
-- An error is returned. If you execute an INSERT statement to insert data into dynamic partitions
, the dynamic partition key columns must be included in the SELECT clause.
```

For more information about the INSERT statements, see [INSERT](#).

4.8.4. JOIN

This topic describes the limits on JOIN statements.

Limits:

- MaxCompute SQL supports the following JOIN operations: {LEFT OUTER|RIGHT OUTER|FULL > OUTER|INNER} JOIN.
- MaxCompute SQL supports a maximum of 128 parallel JOIN operations.

For more information about JOIN statements, see [JOIN](#).

4.8.5. Other limits

This topic describes other limits on MaxCompute SQL.

- MaxCompute SQL supports a maximum of 256 concurrent UNION operations.
- MaxCompute SQL supports a maximum of 256 concurrent INSERT OVERWRITE or INSERT INTO operations.

4.9. Compile and use UDFs

4.9.1. Overview

This topic describes how to write and run a MaxCompute user-defined function (UDF).

MaxCompute supports user-defined scalar functions, user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs).

Note

- UDFs support only Java APIs. To write a UDF program, you can upload UDF code to your project as a resource and execute the required statement to create a UDF.
- This topic provides code examples of user-defined scalar functions, UDAFs, and UDTFs.

4.9.2. UDF example

This topic describes how to create a user-defined scalar function (UDF) and provides an example.

Procedure

1. Write code.

You must develop and compile functions in compliance with the MaxCompute UDF framework.

```
package org.alidata.odps.udf.examples; import com.aliyun.odps.udf.udf;
public final class Lower extends udf { public String evaluate(String s) {
if (s == null) { return null; } return s.toLowerCase();
}
}
```

Name the preceding JAR package *my_lower.jar*.

2. Add resources.

Specify the referenced UDF code before you run a UDF. User code must be added to MaxCompute as resources. Java UDFs must be packaged as JAR packages and added to MaxCompute as JAR resources. The UDF framework automatically loads the JAR packages and runs the UDFs.

Example command to add JAR resources:

```
add jar my_lower.jar;
```

 **Note** If multiple resources have the same name, rename the JAR packages and modify the names of relevant JAR packages in the example command. You can also use the `-f` option to override the existing JAR resources.

3. Register the UDF.

After your JAR package is uploaded, MaxCompute has no information about this UDF. Therefore, you must register a unique function name in MaxCompute and associate the function name with the JAR resource and function.

Example command to register the UDF:

```
create function test_lower as org.alidata.odps.udf.examples.lower using my_lower.jar;
```

Example of the function used in SQL:

```
select test_lower('A') from my_test_table;
```

4.9.3. UDAF example

This topic provides the code example of a user-defined aggregate function (UDAF) for your reference.

UDAFs are registered in the same way as UDFs and are used in the same way as built-in aggregate functions.

UDAF code example:

```
package org.alidata.odps.udf.examples;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
/**
 * project: example_project
 * table: wc_in2
 * partitions: p2=1,p1=2
 * columns: colc,colb,cola
 */
public class UDAFExample extends Aggregator {
    @Override
    public void iterate(Writable arg0, Writable[] arg1) throws UDFException {
        LongWritable result = (LongWritable) arg0;
        for (Writable item : arg1) {
            Text txt = (Text) item;
            result.set(result.get() + txt.getLength());
        }
    }
    @Override
    public void merge(Writable arg0, Writable arg1) throws UDFException {
        LongWritable result = (LongWritable) arg0;
        LongWritable partial = (LongWritable) arg1;
        result.set(result.get() + partial.get());
    }
    @Override
    public Writable newBuffer() {
        return new LongWritable(0L);
    }
    @Override
    public Writable terminate(Writable arg0) throws UDFException {
        return arg0;
    }
}
```

4.9.4. UDTF example

This topic provides an example of user-defined table-valued function (UDTF) code.

UDTFs are registered and used in the similar way to UDFs.

UDTF code example:

```
package org.alidata.odps.udf.examples;
import com.aliyun.odps.udf.udtf;
import com.aliyun.odps.udf.udtfcollector; import com.aliyun.odps.udf.annotation.resolve; import com.
aliyun.odps.udf.udfexception;
// todo define input and output types, e.g., "string,string->string,bigint".
@resolve({"string,bigint->string,bigint"}) public class myudtf extends udtf {
@Override
public void process(object[] args) throws udfexception { string a = (string) args[0];
long b = (long) args[1];
for (string t: a.split("\\s+")) { forward(t, b);
}
}
}
```

4.10. Write and run a MapReduce program

This topic describes how to write and run a MapReduce program.

Prerequisites

- JDK 1.8 is installed.
- The MaxCompute client is configured. For more information, see [Configure the client](#).

Procedure

1. Create input and output tables.

Example:

```
create table wc_in (key string, value string);
create table wc_out (key string, cnt bigint);
```

 **Note** For more information about the statements that are used to create tables, see [Create a table](#).

2. Insert data into the wc_in table.

You can use one of the following methods to insert data:

- Run Tunnel commands to upload data.

The following code shows the data that you want to insert into the table. You must create a kv.txt file on your computer and save the data to the file. In this example, the kv.txt file is saved in D:\.

```
238, val_238
186, val_86
186, val_86
```

Run the following command to upload the data:

```
Tunnel upload D:\kv.txt wc_in;
```

- Execute the following SQL statement to insert the data:

```
INSERT INTO TABLE wc_in VALUES ('238', ' val_238'), ('186', 'val_86'), ('186', 'val_86');
```

3. Write a MaxCompute program and compile it.

- MaxCompute provides the Eclipse plug-in to help you develop MapReduce programs and debug them on your computer.
 - You must create a MaxCompute project in Eclipse. Then, write a MapReduce program in this project. After local debugging succeeds, upload the compiled program (JAR package) to MaxCompute.
4. Add the JAR package as a project resource.

In this example, the JAR package is named word-count-1.0.jar.

```
add jar word-count-1.0.jar;
```

5. Run the JAR command on the MaxCompute client.

```
jar -resources word-count-1.0.jar -classpath /home/resources/word-count-1.0.jar com.taobao.jingfan.wordcount wc_in wc_out;
```

 **Note** If the Java program uses resources, use the `-resources` option to specify the resources.

6. View the command output on the MaxCompute client.

```
select * from wc_out;
```

4.11. Write and run a Graph job

This topic describes how to write and run a Graph job.

Procedure

1. Create input and output tables.

Execute the following statements:

```
create table sssp_in (v bigint, es string);
create table sssp_out (v bigint, l bigint);
```

 **Note** For more information about the statements that are used to create tables, see [Create a table](#).

2. Upload data to the sssp_in table.

We recommend that you create an sssp.txt file that contains the following data and save the file to a local directory, such as D:\:

```
1 2:2,3:1,4:4
2 1:2,3:2,4:1
3 1:1,2:2,5:1
4 1:4,2:1,5:1
5 3:1,4:1
```

Run the Tunnel command to upload data to the sssp_in table. Use spaces as the delimiter between columns.

```
tunnel u -fd " " D:\sssp.txt sssp_in;
```

3. Compile a Single Source Shortest Path (SSSP) example.

Compile and debug SSSP algorithm examples in local mode based on the Graph development process. For more information, see [SSSP](#).

Note In this example, the code is packaged as *odps-graph-example-sssp.jar*. You need only to package the SSSP code. You do not need to package the SDK into this JAR package.

4. Add a JAR resource.

Examples:

```
add jar $LOCAL_JAR_PATH/odps-graph-example-sssp.jar odps-graph-example-sssp.jar
```

Note For more information about how to add resources, see [Add a resource](#).

5. Run the SSSP.

Examples:

```
jar -libjars odps-graph-example-sssp.jar -classpath $LOCAL_JAR_PATH/odps-graph-example-sssp.jar com.aliyun.odps.graph.examples.sssp 1 sssp_in sssp_out;
```

Note

The MaxCompute client provides a jar command to run MaxCompute Graph jobs. You can use this command for Graph jobs in the same way as MapReduce jobs.

The commands used to run Graph jobs list the instance ID, execution progress, and result summary of the job.

Example:

```
ID = 20130730160742915g*****
2013-07-31 00:18:36 SUCCESS
Summary:
Graph Input/Output
Total input bytes=211
Total input records=5
Total output bytes=161
Total output records=5
graph_input_[bsp.sssp_in]_bytes=211
graph_input_[bsp.sssp_in]_records=5
graph_output_[bsp.sssp_out]_bytes=161
graph_output_[bsp.sssp_out]_records=5
Graph Statistics
Total edges=14
Total halted vertices=5
Total sent messages=28
Total supersteps=4
Total vertices=5
Total workers=1
Graph Timers
Average superstep time (milliseconds)=7
Load time (milliseconds)=8
Max superstep time (milliseconds) =14
Max time superstep=0
Min superstep time (milliseconds)=5
Min time superstep=2
Setup time (milliseconds)=277
Shutdown time (milliseconds)=20
Total superstep time (milliseconds)=30
Total time (milliseconds)=344
OK
```

4.12. Use Logview to view job running information

This topic describes how to use Logview to view job running information.

Logview is a tool that you can use to view and debug jobs after you submit the jobs to MaxCompute. You can use Logview to view the following information of a job:

- Running status of tasks
- Running results of tasks
- Details of each task and the progress of each step

Note When you deploy MaxCompute, Logview is deployed by default.

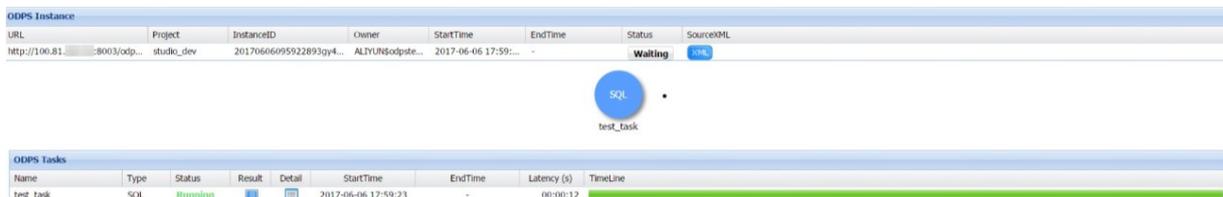
After you submit a job to MaxCompute, the system generates a Logview URL. You can enter the Logview URL in the search engine and press Enter to view job information.

```
odps@ test_workshop001>select * from result_test;
ID = 20190917055240226ga6e5mim
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcomp
m/api&p=test_workshop001&i=20190917055240226ga6e5mim&token=YkxhUzJQ
1NIamRWnDBBU3UJPSxPRFBTX09CTzoxMDc50TI20Dk20Tk5NDIxLDE1NjkzMDQzNjAs
nQi01t7IkFjdGlvbiI6WyJvZHBzO1JlYWQiXSwiRWZmZWN0IjoiQWxsY3ciLCJSZXNv
3M6b2Rwcz0nByb2p1Y3RzL3Rlc3Rfd29ya3Nob3AwMDEvaW5zdGFuY2UzLzIwMTkw
jI22E2ZTUtaW0iXX1dLCJWZXJzaW9uIjoMSJ9
Job Queueing...
Summary:
+-----+-----+
| education | num |
+-----+-----+
```

Note The Logview page of each job is valid for seven days.

UI elements

This section describes elements on the Logview UI.



The Logview page consists of two sections:

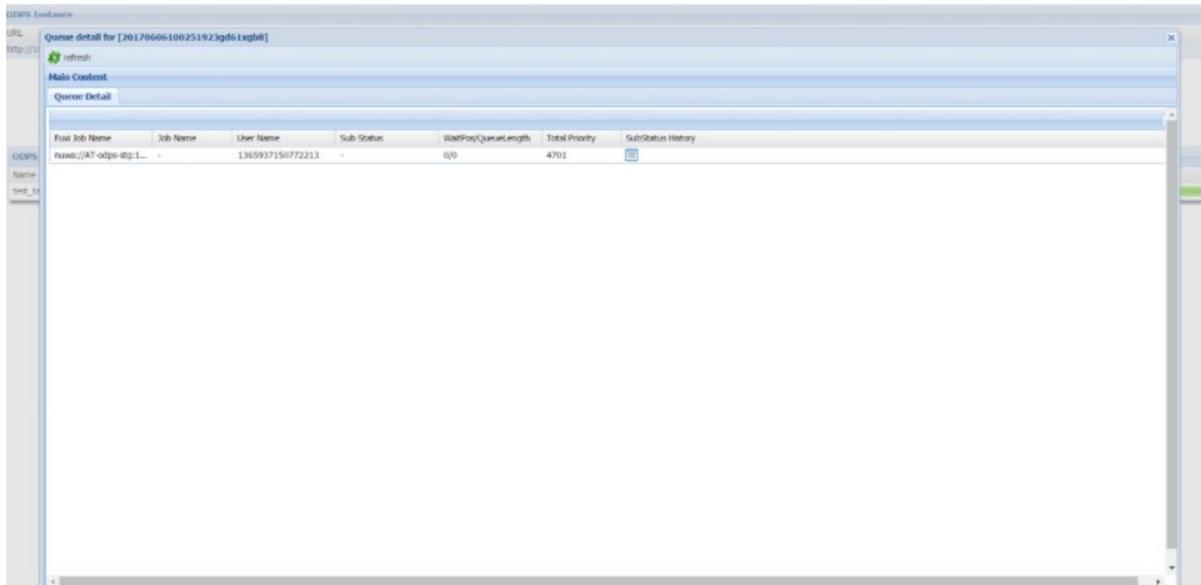
- ODPS Instance
- ODPS Tasks

The ODPS Instance section shows the information of the instance that corresponds to the submitted SQL job. The information includes URL, Project, InstanceID, Owner, Start Time, End Time, and Status.

- You can click the value in the **Status** column to view queue information. Valid values of Status:
 - Waiting: The job is being processed in MaxCompute and has not been submitted to Job Scheduler.
 - Waiting List : n: The job has been submitted to Job Scheduler and is queued. n indicates the order number of the job in the queue.
 - Running: The job is running in Job Scheduler.

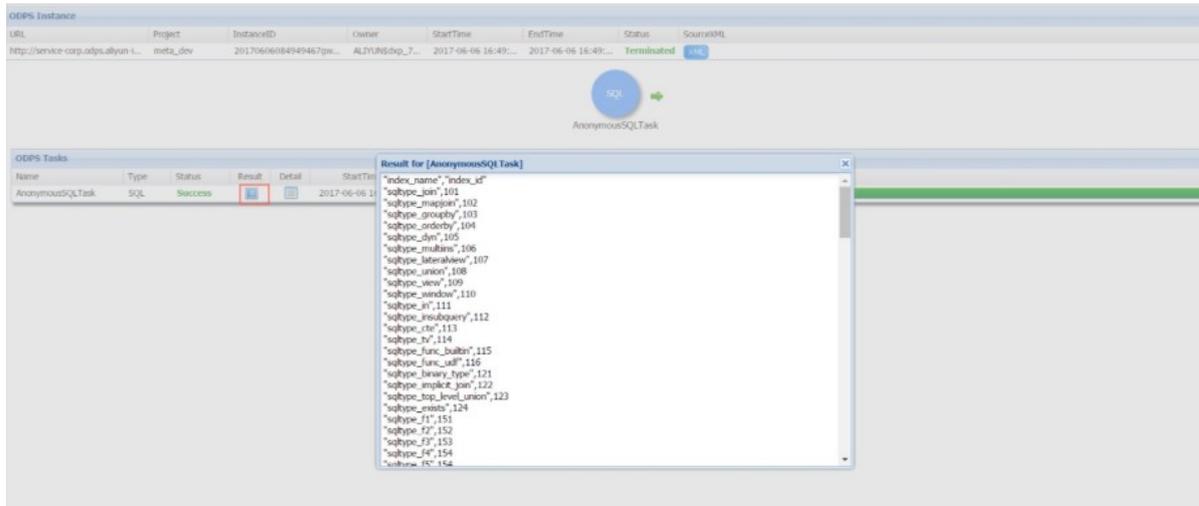
 **Note** If the value of Status is Terminated, the job is complete and has no queue information.

- After you click the value of Status, the following queue information is displayed:
 - Sub Status: the current sub-status information.
 - WaitPos: the position of the job in the queue. The value 0 indicates that the job is running. The value - indicates that the job has not been submitted to Job Scheduler.
 - QueueLength: the total queue length in Job Scheduler.
 - Total Priority: the running priority assigned by the system.
 - SubStatus History: You can click the icon in this column to view the status history. This includes the status code, status description, start time, and duration of a state. The information is unavailable in some versions.

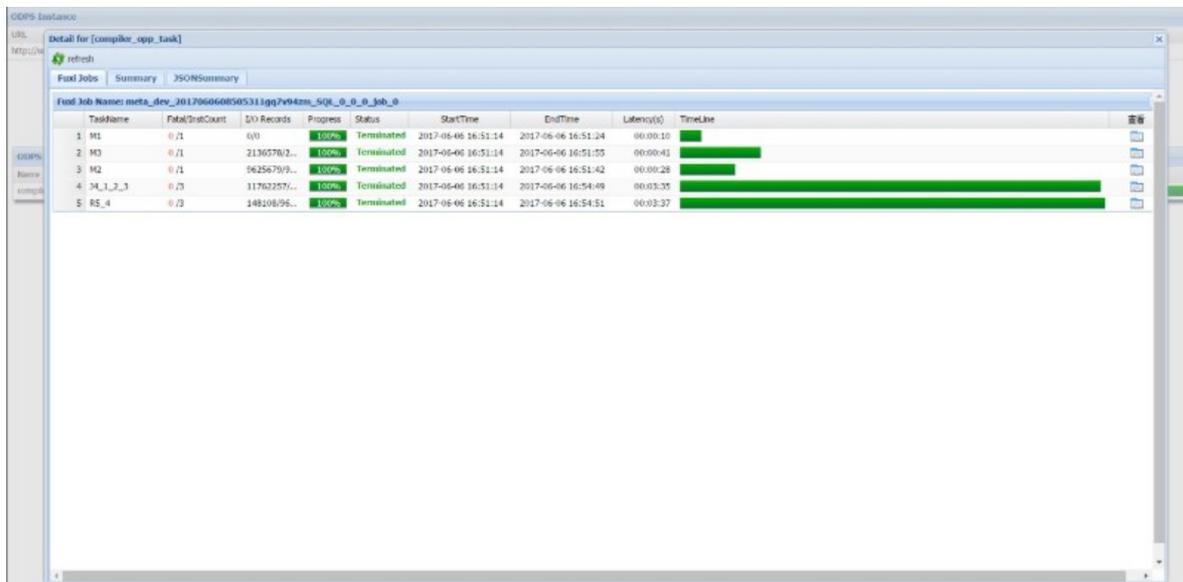


The ODPS Tasks section shows the information of the task that corresponds to the instance. The information includes Name, Type, Status, Result, Detail, StartTime, EndTime, Latency (s), and TimeLine. Like on other pages, Latency (s) indicates the running duration.

- Result: After a job is complete, you can click the icon in the Result column to view execution results. The following figure shows the execution results of a SELECT statement.



- Detail: You can click the icon in the **Detail** column to view running details. You can view the details of both running and completed jobs.



- In the dialog box that shows details about a MaxCompute job, you can view the following information:
 - A MaxCompute job consists of one or more Fuxi jobs. For example, if a complex SQL job is submitted, MaxCompute automatically submits multiple Fuxi jobs to Job Scheduler.
 - Each Fuxi job consists of one or more Fuxi tasks. For example, a simple MapReduce job generates two Fuxi tasks: map task (M1) and reduce task (R2). If an SQL job is complex, multiple Fuxi tasks may be generated.
 - You can view the name of a Fuxi task. In most cases, a task name consists of letters and digits. A letter represents the type of a task, such as M for a map task. The digits that follow the letter represent the ID and dependencies of a task. For example, R5_4 indicates that the reduce task can be executed only after the J4 task is complete. J4_1_2_3 indicates that the join task can be executed only after the M1, M2, and M3 tasks are complete.
 - I/O Records indicates the numbers of the input and output records of the Fuxi task.
- To view the information of the instance that corresponds to a Fuxi task, you can click the icon in the Show Detail column of the Fuxi task or double-click the Fuxi task.

Note Each Fuxi task consists of one or more Fuxi instances. If the amount of input data for a Fuxi task increases, MaxCompute starts more nodes for the task to process the data. A node is equivalent to a Fuxi instance.

In the lower part of the dialog box, the status information about Fuxi instances at different stages is displayed in groups. For example, you can click the Failed tab to view the nodes where errors occurred. You can click the icon in the StdOut column to view standard output information. You can also click the icon in the StdErr column to view standard error information.

Note To-be-displayed information written in the submitted MaxCompute job is also displayed in the standard output information and standard error information.

Use Logview to handle issues

Tasks with errors: If an error occurs during a task, find the task and click the icon in the Result column in the lower part of the Logview page to view the error information. You can also click the icon in the StdErr column of a Fuxi instance in the details dialog box to view the error information of the instance.

Data skew: The long tail of one or more Fuxi instances may decelerate the execution of a Fuxi task. The long tail is caused by uneven data distribution in a task. After the task is complete, you can view the running results on the Summary tab of the details dialog box. The following output provides an example of the running results.

```
output records:
R2_1_Stg1: 199998999 (min: 22552459, max: 177446540, avg: 99999499)
```

If the difference between the min and max values is large, data skew occurs. For example, if a specific value appears more often than other values in a column, data skew occurs when you execute a JOIN operation based on this column.

4.13. Use Logview V2.0 to view job running information

This topic describes the entry and features of Logview V2.0. You can use Logview V2.0 to view job running information.

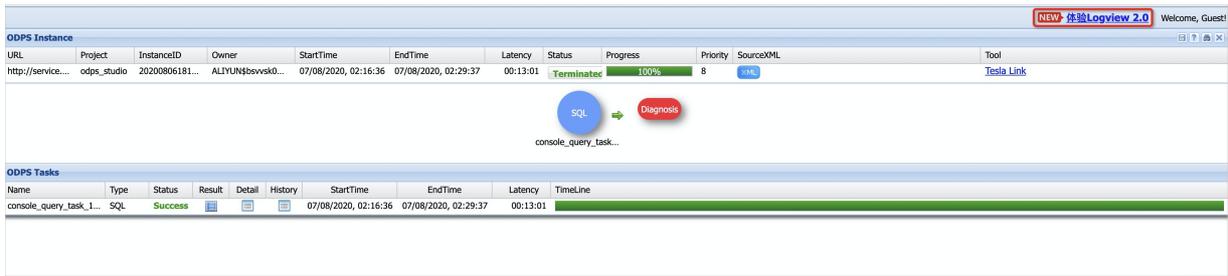
Overview

Logview V2.0 provides a newly designed UI, increases the loading speed, and delivers the following new features:

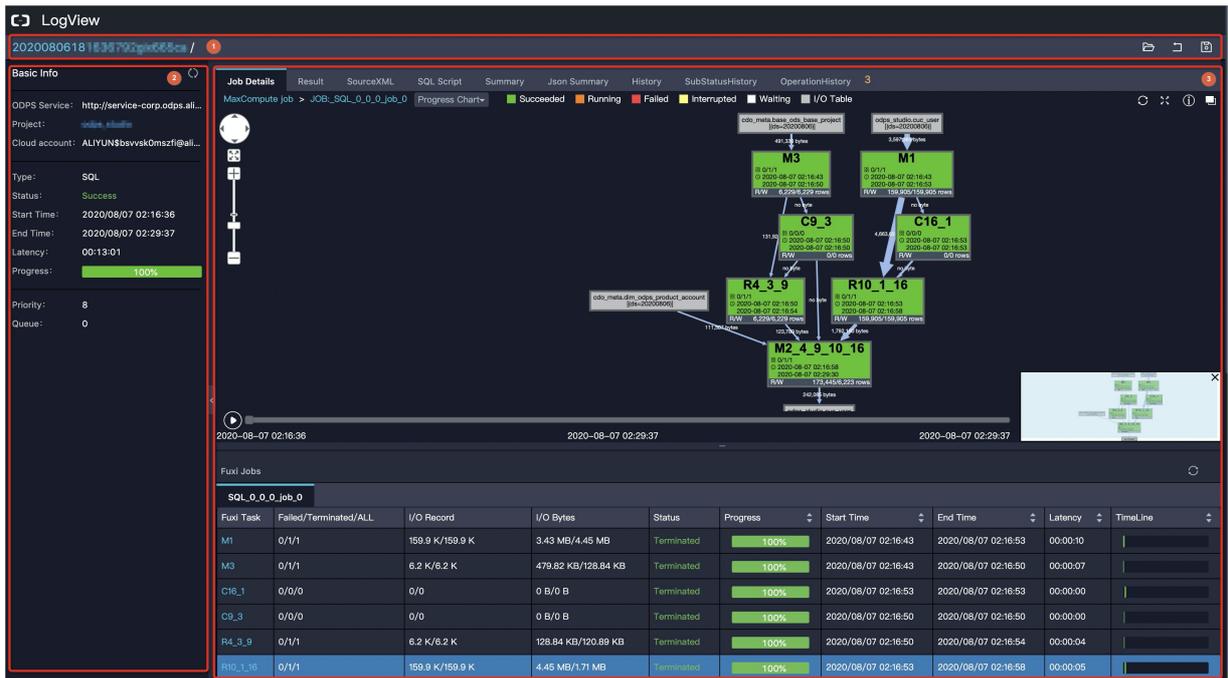
- Provides an interactive directed acyclic graph (DAG) to display the logical architecture of job processing. You can also view the operators of a job.
- Supports job execution playback.
- Allows you to view memory usage and CPU utilization by using Fuxi Sensor.

Entry

After you use the MaxCompute client (odpscmd) to submit a job, the system generates a Logview URL. Enter the Logview URL in the search engine and press Enter. On the Logview UI, click **Go to Logview V2.0**.



Logview V2.0 page



No.	Section
①	The title and function section.
②	The Basic Info section.
③	The job details section.

Title and function section

This section shows the job ID and job name. The job ID uniquely identifies a MaxCompute job. The job ID is generated when you submit the job. The job name is displayed only if the job is submitted by using SDKs. You can also click the icons on the right of this section to perform operations.

Icon	Description
	Open the Logview_detail.txt file that contains job details. The file is saved on your computer.
	Return to the original Logview UI.

Icon	Description
	Save the job details as a file to your computer.

Basic Info

This section shows the basic information about a job.

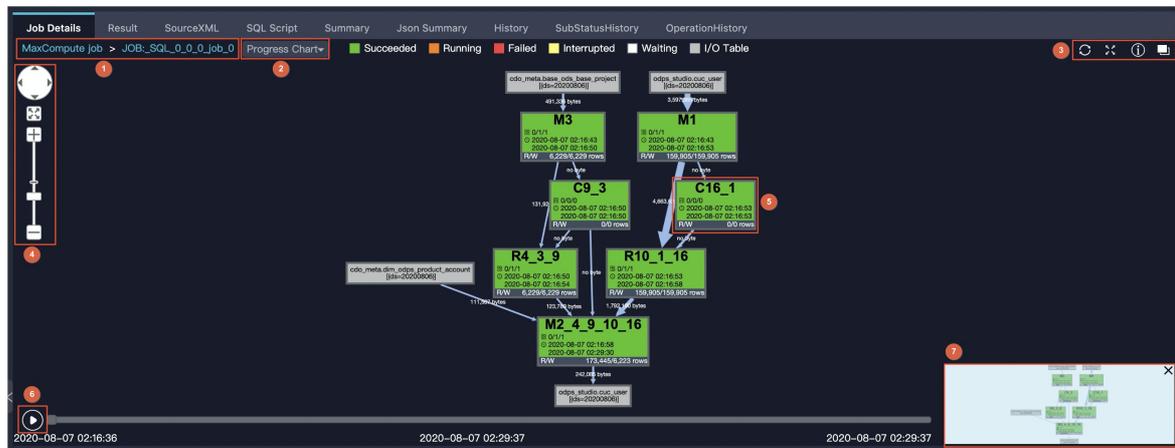
Parameter	Description
MaxCompute Service	The endpoint of MaxCompute on which the job is running.
Project	The name of the MaxCompute project to which the job belongs.
Cloud account	The Alibaba Cloud account that is used to submit the job.
Type	The type of the job. Valid values: SQL, SQLRT, LOT, XLib, CUPID, AlgoTask, and Graph.
Status	The status of the job. Valid values: <ul style="list-style-type: none"> • Success: The job succeeds. • Failed: The job fails. • Canceled: The job is canceled. • Waiting: The job is being processed in MaxCompute but is not submitted to Job Scheduler. • Running: The job is being processed in Job Scheduler. • Terminated: The job is complete.
Start Time	The time when the job is submitted.
End Time	The time when the job is complete.
Latency	The period during which the job is executed.
Progress	The progress of the job.
Priority	The priority of the job.
Queue	The position of the job in the queue in the resource quota group.

Job details

In the job details section, you can query details about a job. This section consists of the following tabs:

- **Job Details**
 - Progress chart

In the upper part of the **Job Details** tab, the progress chart of a job is displayed. The progress chart shows the subtask dependencies from three dimensions: Fuxi jobs, Fuxi tasks, and operators. It also provides a series of tools to help you troubleshoot issues. The following figure shows the upper part of the Job Details tab.



No.	Description
①	The breadcrumb navigation that is used to switch Fuxi jobs. JOB:_SQL_0_0_0_job_0 is the name of a Fuxi job.
②	The troubleshooting tool. You can use Progress Chart, Input Heat Chart, Output Heat Chart, TaskTime Heat Chart, and InstanceTime Heat Chart for troubleshooting.
③	You can click the icon to refresh the job status and the icon to zoom in or out the progress chart. You can also click the icon to obtain MaxCompute Studio documentation and the icon to switch to the upper level of the job.
④	The zoom tool.
⑤	<p>The Fuxi task. A MaxCompute job consists of one or more Fuxi jobs. Each Fuxi job consists of one or more Fuxi tasks. Each Fuxi task consists of one or more Fuxi instances. If the amount of input data increases, MaxCompute starts more nodes for each task to process the data. A node is equivalent to a Fuxi instance. For example, a simple MapReduce job generates two Fuxi tasks: map task (M1) and reduce task (R2). If an SQL statement is complex, multiple Fuxi tasks may be generated.</p> <p>You can view the name of each Fuxi task on the execution page. For example, M1 indicates a map task. The 3 and 9 fields in R4_3_9 indicate that the map task can be executed only after M3 and C9_3 are complete. Similarly, M2_4_9_10_16 indicates that the M2 task can be executed only after R4_3_9, C9_3, R10_1_16, and C16_1 are complete. R/W indicates the numbers of rows that the task reads and writes.</p> <p>Click or right-click a task to view operator dependencies and operator graphs of the task.</p> <p>Logview V2.0 provides table dependencies so that you can quickly view the input and output tables.</p>
⑥	The Fuxi task playback. You can click the icon to start or stop the playback. You can drag the progress bar to play at a specific time. The start time and end time are displayed on the sides of the progress bar. The playing time is displayed in the middle.

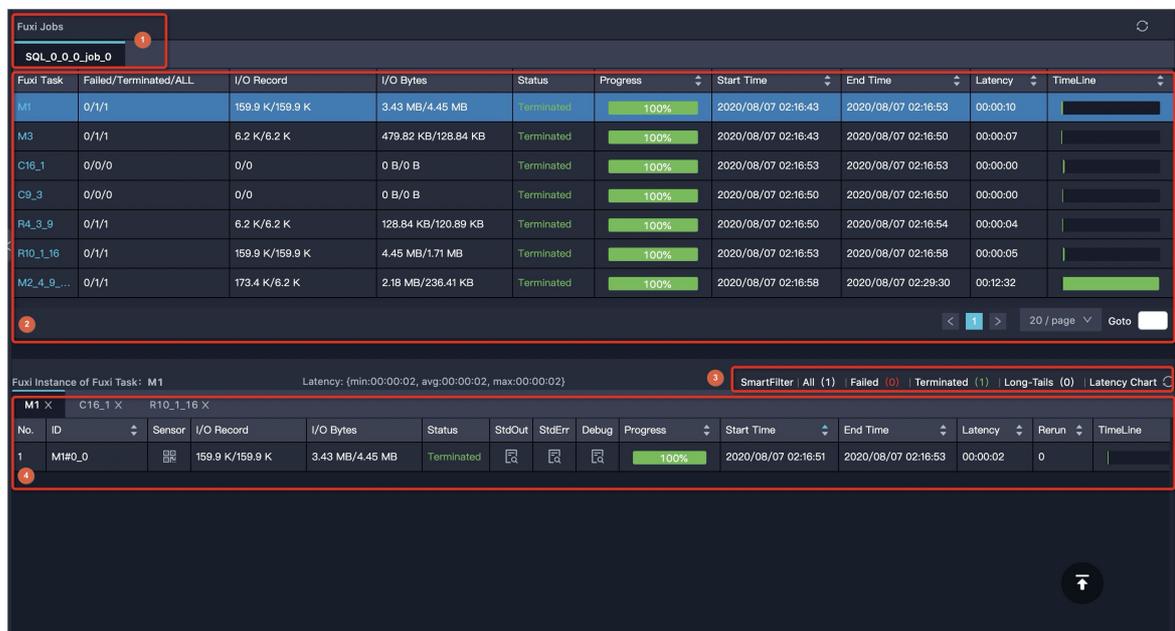
No.	Description
⑦	EagleEye.

Note

- The playback feature is not applicable to Fuxi tasks that are in the Running state.
- An AlgoTask job, such as a Machine Learning Platform for AI (PAI) job, contains only one Fuxi task. Therefore, no progress charts are provided for these jobs.
- For non-SQL jobs, only Fuxi jobs and Fuxi tasks are displayed.
- If only one Fuxi job exists, the progress chart shows the dependencies among Fuxi tasks. If multiple Fuxi jobs exist, the progress chart shows the dependencies among the Fuxi jobs.

o Running status of jobs

In the lower part of the Job Details tab, the detailed running information about the job is displayed. The following figure shows the lower part of the Job Details tab.

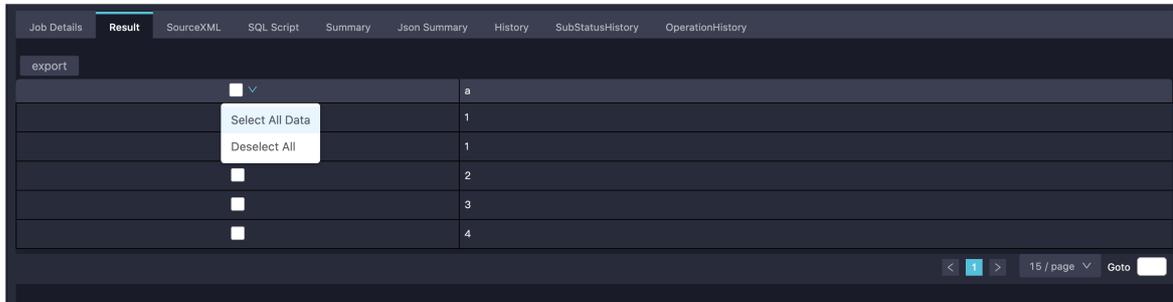


No.	Description
①	The Fuxi Jobs tab. You can switch Fuxi jobs on this tab.
②	The details about Fuxi tasks of the Fuxi job. Click a Fuxi task to display information about the Fuxi instance of this task. By default, information about the Fuxi instance of the first Fuxi task for the first Fuxi job is displayed.
③	Logview divides instances into groups based on their status. You can click the number next to Failed to query information about faulty nodes.
④	Fuxi Sensor. This feature is provided only for AlgoTask jobs, such as PAI jobs. You can use Fuxi Sensor to view the memory usage and CPU utilization of Fuxi instances.
⑤	StdOut and StdErr. You can view output messages, error messages, and information to be displayed. You can also download the information.
⑥	Debug. You can debug and troubleshoot errors.

● **Result**

This tab shows the result of a job. If a job failed, this tab is displayed and shows the cause of the failure. The system displays data in plain text or a table based on the response format specified in MaxCompute. You can log on to the MaxCompute client and set `odps.sql.select.output.format` to specify the format. If you set this parameter to `HumanReadable`, the result is displayed in plain text. Otherwise, the result is displayed in the CSV format.

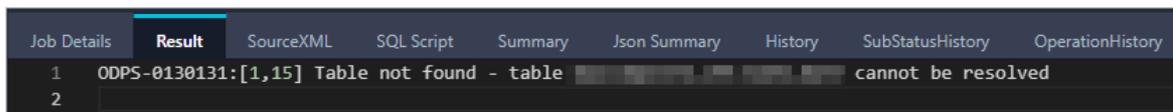
○ Table



You can perform the following operations on this tab:

- Select a row to export its data.
- Select the table heading to export data from the current page.
- In the table heading, click the icon and select **Select All Data** or **Deselect All** as needed.
- Click **export** to export data in the CSV format. CSV is a file name extension. You can use a text file or Sublime Text to open CSV files.

○ Plain text



● **SourceXML**

This tab shows the source XML information about the job that is submitted to MaxCompute.

● **SQL Script**

This tab shows the SQL scripts of the job that is submitted to MaxCompute.

● **Summary**

This tab shows the overall running information about the job that is submitted to MaxCompute.

● **Json Summary**

This tab shows the overall running information about the job in the JSON format.

● **History**

This tab shows the historical information about a Fuxi instance if the instance is executed multiple times.

● **SubStatusHistory**

This tab shows the detailed information about a job, including the status code, status description, start time, duration, and end time.

Fuxi Sensor

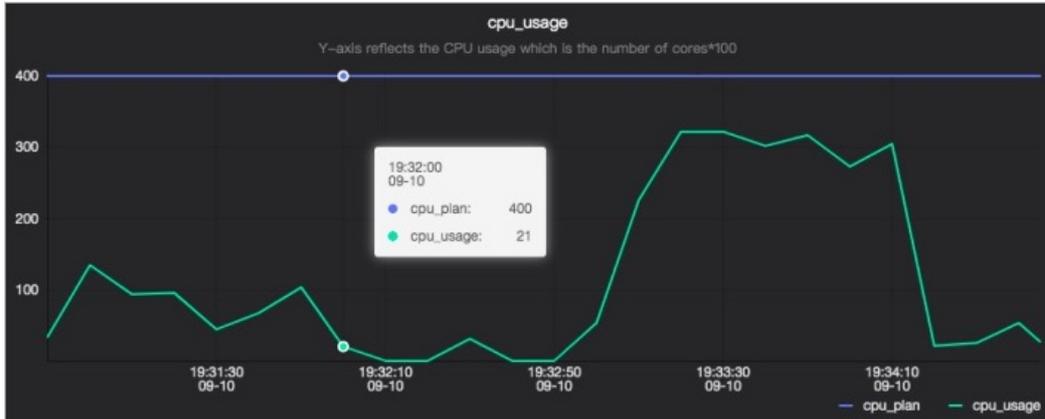
Fuxi Sensor is a resource view that shows a MaxCompute job from all dimensions. You can use Fuxi Sensor to view the memory usage and CPU utilization of a Fuxi instance. You can also use Fuxi Sensor to locate job issues and analyze running performance. For example, you can use Fuxi Sensor in the following scenarios:

- If out-of-memory (OOM) occurs, analyze the amount of memory used.
- Compare the numbers of requested and used resources to optimize the resource request process.

For example, you can use Fuxi Sensor to view the resource usage of a Fuxi instance.

- CPU utilization

The `cpu_usage` chart has two lines. One indicates the number of requested CPUs (`cpu_plan`), and the other one indicates the number of used CPUs (`cpu_usage`). In the y-axis, 400 indicates four processors.

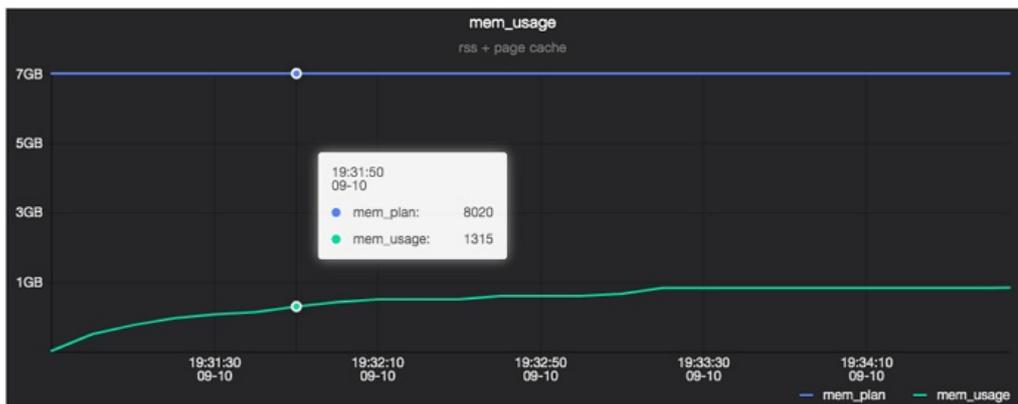


- Memory usage

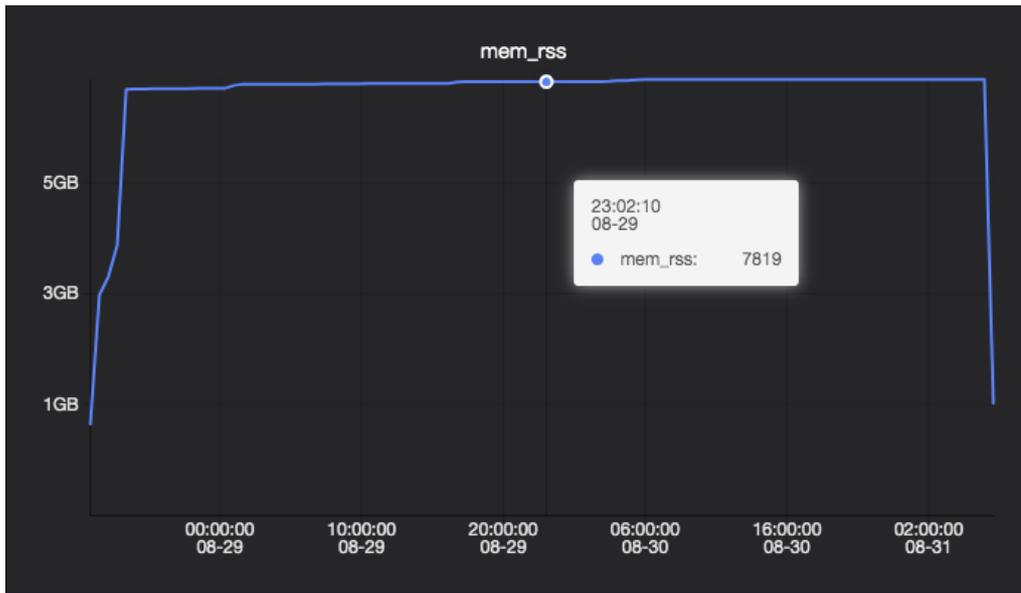
The `mem_usage` chart has two lines. One indicates the number of requested memory resources (`mem_plan`), and the other one indicates the number of used memory resources (`mem_usage`).

`mem_usage` contains Resident Set Size (RSS) and PageCache. RSS indicates the memory that is allocated after kernel page faults occur. This applies when you call Malloc to request memory by using non-file mappings. If the memory is insufficient, RSS cannot be reclaimed. PageCache is the memory occupied by the kernel to cache files that are required by the read and write requests, such as log files. If the memory is insufficient, PageCache can be reclaimed.

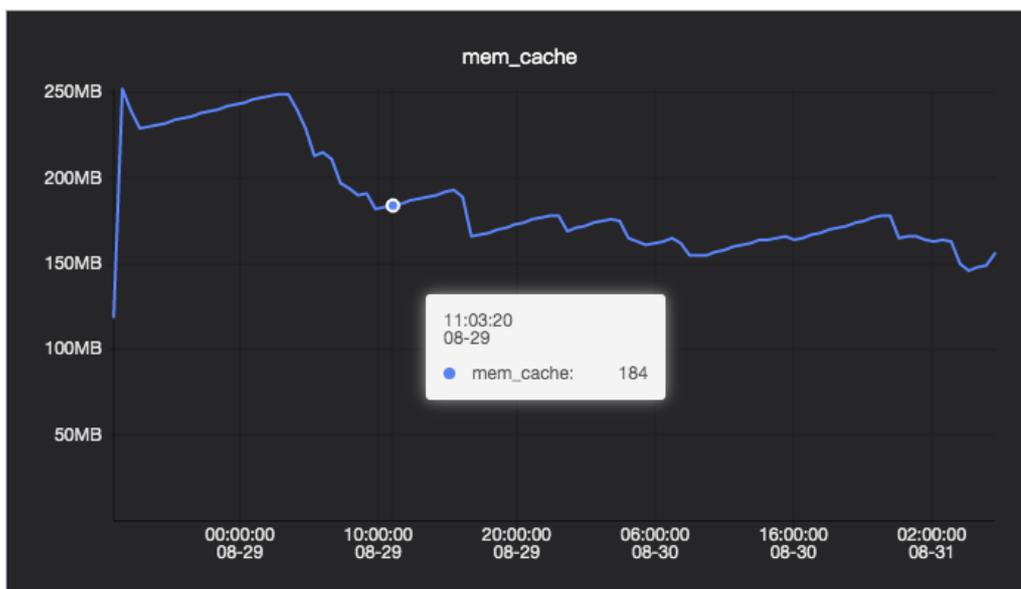
- Memory details



- o RSS usage



- o PageCache usage



5. Basic concepts and common commands

5.1. Terms

This topic introduces the basic terms of MaxCompute.

Project

A basic organizational unit of MaxCompute. Similar to a database or schema in a traditional database system, a project is used to isolate multiple users and control access requests. A user can have permissions on multiple projects. After authorization, you can access objects, such as tables, resources, functions, and instances in a project from another project.

You can run the `use project` command to access a project. Example:

```
use al_project
-- Access a project named al_project.
```

 **Note** After you run the preceding command, you access a project named `al_project` and manage objects that belong to this project, regardless of which project you are in.

Table

A data storage unit of MaxCompute. Logically, a table is a two-dimensional structure that consists of rows and columns, in which each row represents a record and each column represents the values of the same data type. One record can contain one or more columns. The column names and types constitute the schema of this table. All data in MaxCompute is stored in tables. Table columns support all data types in MaxCompute. Tables are the input and output objects of all MaxCompute computing tasks. You can create a table, delete a table, and import data into a table.

You can define partitions for a table to process data more efficiently. You can specify some fields in the table as partition key columns. In most cases, partitions within a table are analogous to directories within a file system. Each value of a partition key column is called a partition in MaxCompute. You can group multiple fields of a table to a single partition to create a multi-level partition. Multi-level partitions are analogous to multi-level directories. If you specify the name of the partition that you want to access, MaxCompute scans only the specified partition. This makes data processing more efficient and reduces costs.

Data type

The values in a column of a MaxCompute table must be of a specific data type. The following table describes the basic data types that are supported by MaxCompute and their valid values.

Basic data types

Data type	New in MaxCompute V2.0	Constant	Description	Valid value
TINYINT	Yes	1Y,-127Y	The 8-bit signed integer type.	-128 to 127
SMALLINT	Yes	32767S,-100S	The 16-bit signed integer type.	-32768 to 32767

Data type	New in MaxCompute V2.0	Constant	Description	Valid value
INT	Yes	1000,-15645787	The 32-bit signed integer type.	-2^{31} to $2^{31}-1$
BIGINT	No	100000000000L,-1L	The 64-bit signed integer type.	$-2^{63}+1$ to $2^{63}-1$
STRING	No	"abc",'bcd',"alibaba", 'inc'	The string, which supports UTF-8 encoding. The character behavior encoded in other formats is not defined.	The size of all values in a column of the STRING type cannot exceed 8 MB.
FLOAT	Yes	None	The 32-bit binary floating point type.	N/A
BOOLEAN	No	True,False	The BOOLEAN type.	True or False
DOUBLE	No	3.1415926 1E+7	The 64-bit binary floating point type.	-1.0×10^{308} to 1.0×10^{308}
DATETIME	No	Datetime '2017-11-11 00:00:00'	The DATETIME type. The standard system time is UTC+8.	0001-01-01 00:00:00 000 to 9999- 12-31 23:59:59 999
DECIMAL	No	3.5BD, 9999999999.99999 99BD	The precise numeric type based on the decimal system.	Integer: $-10^{36}+1$ to $10^{36}-1$ Decimal places: round to 10^{-18}
VARCHAR	Yes	None	The variable-length character type, in which n specifies the length.	1 to 65535
BINARY	Yes	None	The binary data type.	The size of all values in a column of the BINARY type cannot exceed 8 MB.
TIMESTAMP	Yes	Timestamp '2017-11-11 00:00:00.123456789'	The TIMESTAMP type. The timestamp is independent of time zones.	0001-01-01 00:00:00 000000000 to 9999- 12-31 23:59:59 999999999

If you want to use the new data types in MaxCompute V2.0, you must first add one of the following flags to enable the new data type edition: `set odps.sql.type.system.odps2=true;` at the session level or `setproject odps.sql.type.system.odps2=true;` at the project level. Otherwise, the following error may occur: `xxxx type is not enabled in current mode`. The data types listed in the preceding table can contain NULL values.

 **Note** Only lowercase letters can be used in the preceding flags.

Take note of the following points when you use the new data types in MaxCompute V2.0:

- If you add the `set odps.sql.type.system.odps2=true;` flag, the following impacts occur:
 - The semantics of the INT keyword changes. INT in an SQL statement indicates a 32-bit integer.
 - The semantics of an integer constant changes. `Select 1 + a;` is used in this example.
 - If the new data type edition is disabled, the integer constant is processed as the BIGINT type. If the length of the constant exceeds the range for the BIGINT type, the integer constant is processed as the DOUBLE type.
 - If the new data type edition is enabled, the integer constant is processed as the INT type. In this example, the constant 1 is a 32-bit integer.
 - If the integer constant is processed as the INT type, inconsistency may occur in function prototypes during subsequent operations. In addition, the actions of peripheral tools and subsequent operations might be changed due to the tables that contain new data types and that are generated after data is written to a disk.
 - The rules for implicit conversions of data types change.

If the new data type edition is enabled, some implicit type conversions may be disabled. For example, if the data type is converted from STRING to BIGINT, from STRING to DATETIME, from DOUBLE to BIGINT, from DECIMAL to DOUBLE, or from DECIMAL to BIGINT, precision may be reduced or errors may occur. In this case, you can use the CAST function to force the data type conversion.

Implicit type conversions greatly affect functions and INSERT statements. For example, an INSERT statement can be executed if the new data type edition is disabled, but an error is returned if the data type edition is enabled.
 - If the new data type edition is disabled, some operations and built-in functions that use data of new data types as parameters and return values are ignored. These operations and functions can be used after the new data type edition is enabled.
 - Some built-in functions can be used only after the new data type edition is enabled. The built-in functions include the functions that return the values of the INT type, such as YEAR, QUARTER, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND, NANOSECOND, DAYOFMONTH, and WEEKOFYEAR. The parameters of these functions can be of the INT type, and the BIGINT overload is used for these functions. These functions can be implemented by using the DATEPART built-in function.
 - The resolution of user-defined functions (UDFs) changes.
 - The parsing of the BIGINT keyword changes.
 - The data types supported by partition key columns change.
 - If the new data type edition is disabled, the data type of a partition key column can only be STRING.
 - If the new data type edition is enabled, the data type of a partition key column can be STRING, VARCHAR, CHAR, TINYINT, SMALLINT, INT, or BIGINT.
 - If the new data type edition is disabled, partition fields in INSERT operations are processed as the STRING type.
 - The execution rules of LIMIT statements in SET operations change.

The `Select * from t1 union all select * from t2 limit 10;` statement is used in this example.

 - If the new data type edition is disabled, the preceding statement is expressed as `Select * from t1 union all select * from (select * from t2 limit 10) t2;` .
 - If the new data type edition is enabled, the preceding statement is expressed as `Select * from (select * from t1 union all select * from t2) limit 10;` .

These changes also apply to UNION, INTERSECT, EXCEPT, ORDER BY, DISTRIBUTE BY, SORT BY, and CLUSTER BY.

- The parsing of data types in IN expressions changes.
 - The `a in (1, 2, 3)` expression is used in this example.
 - If the new data type edition is disabled, all the values enclosed in the parentheses () must be of the same type.
 - If the new data type edition is enabled, all the values enclosed in the parentheses () are implicitly converted into the same type.
- If a constant of the INT type exceeds the range for the INT type, the constant is processed as the BIGINT type. If the constant exceeds the range for the BIGINT type, the constant is processed as the DOUBLE type. If `odps.sql.type.system.odps2` is not set to true, MaxCompute retains the conversion and notifies you that the INT data is processed as the BIGINT type. If `odps.sql.type.system.odps2` is not set to true in your scripts, we recommend that you rewrite your scripts and convert these types into BIGINT to prevent confusion.
- VARCHAR constants can be implicitly converted into STRING constants.
- STRING constants can be combined. For example, `'abc'` and `'xyz'` can be combined as `'abcxyz'`. Different parts can be written in different rows.
- Time values do not include the millisecond component. You can add `-dfp` to Tunnel commands to display milliseconds in the time values.

MaxCompute supports complex data types. The following table lists their definitions and constructors.

Complex data types

Data type	Definition	Constructor
ARRAY	<pre>array< int >; array< struct< a:int, b:string >></pre>	<pre>array(1, 2, 3); array(array(1, 2); array(3, 4))</pre>
MAP	<pre>map< string, string >; map< smallint, array< string>></pre>	<pre>map("k1", "v1", "k2", "v2"); map(1S, array('a', 'b'), 2S, array('x', 'y'))</pre>
STRUCT	<pre>struct< x:int, y:int>; struct< field1:bigint, field2:array< int>, field3:map< int, int>></pre>	<pre>named_struct('x', 1, 'y', 2); named_struct('field1', 100L, 'field2', array(1, 2), 'field3', map(1, 100, 2, 200))</pre>

If PyODPS is used, you can use one of the following methods to enable the new data type edition:

- Execute `o.execute_sql('set odps.sql.type.system.odps2=true;query_sql', hints={"odps.sql.submit.mode": "script"})` to enable the new data type edition.
- Use DataFrame to enable the new data type edition. For example, you can set the hints parameter to specify an immediately executed method, such as `persist`, `execute`, or `to_pandas`. The settings in the following figure are valid only for a single job.

```
from odps.df import DataFrame
users = DataFrame(o.get_table('odps2_test'))
users.persist('copy_test', hints={'odps.sql.type.system.odps2': 'true'})
```

- If you use DataFrame to enable the new data type edition and want the settings to take effect globally, set

```
options.sql.use_odps2_extension
```

 to True.

Resource

A unique concept in MaxCompute. To implement UDF and MapReduce features, you must use resources.

- **MaxCompute SQL UDFs:** After you write a UDF, you must package it into a JAR file and upload the package to MaxCompute as a resource. When you run the UDF, MaxCompute automatically downloads the JAR file and obtains the code to run the UDF. JAR files are a type of MaxCompute resource. When you upload a JAR file, a resource is created in MaxCompute.
- **MaxCompute MapReduce:** After you write a MapReduce program, you must package it into a JAR file and upload the file to MaxCompute as a resource. When you run a MapReduce job, the MapReduce framework automatically downloads the JAR file and obtains the code to run the MapReduce job.

Note

- Some limits are imposed on how MaxCompute UDFs and MapReduce access resources. For more information, see [Limits](#).
- You can also upload tables or text files to MaxCompute as different types of resources. You can read or use these resources when you run UDFs or MapReduce jobs. MaxCompute provides APIs for you to read and use resources. For more information, see [UDF description](#).

MaxCompute supports the following resource types:

- File
- Table: tables in MaxCompute.
- JAR: compiled JAR files.
- Archive: compressed files identified by the resource name extension. Supported file types include .zip, .tgz, .tar, and .jar.
- Py: Python scripts used by Python UDFs.

Note For more information about resource operations, see [Resource operations](#).

UDF

MaxCompute provides SQL computing capabilities. You can use the built-in functions in MaxCompute SQL statements to complete some computing and counting tasks. If these built-in functions do not meet your business requirements, you can use the Java APIs that MaxCompute provides to develop UDFs. UDFs are classified into user-defined scalar-valued functions, user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs).

After you write the UDF code, you must package the code into a JAR file, upload the file to MaxCompute as a resource, and then register this UDF in MaxCompute. To use a UDF in MaxCompute, you need to only specify its name and parameters in an SQL statement as you do when you use built-in functions of MaxCompute.

Note For more information about UDF operations, see [Function operations](#).

Task

A basic computing unit of MaxCompute. SQL and MapReduce features are all implemented as tasks. MaxCompute parses most of the tasks that you submit, especially computing tasks, such as SQL DML statements and MapReduce tasks. Then, MaxCompute generates a task execution plan based on the parsing results.

An execution plan consists of several dependent stages. An execution plan can be logically defined as a directed graph. Vertices of the graph represent stages, and edges of the graph represent dependencies between stages. MaxCompute executes the stages based on the dependencies in the graph (execution plan). A stage has multiple processes, also known as workers. The workers in each stage work together to complete data computations for the stage. Different workers in a stage process different data, but they all use the same execution logic.

When you run a computing task, the task is converted into an instance. You can perform operations on this instance. For example, you can obtain the status of the instance and terminate the instance.

Some MaxCompute tasks, such as SQL DDL statements, are not computing tasks. These tasks read and modify only metadata in MaxCompute. MaxCompute cannot generate execution plans for these tasks.

 **Notice** MaxCompute does not convert all requests into tasks. For example, project, resource, UDF, and instance operations are not executed as tasks.

Instance

Some MaxCompute tasks are converted into instances during the execution process. An instance has two stages: Running and Terminated. Instances that are in the Running stage are also in the Running state. Instances that are in the Terminated stage can be in the Success, Failed, or Canceled state. You can query or modify the status of a running task by using the instance ID provided by MaxCompute. Example:

```
status <instance_id>;
-- Query the status of an instance.
kill <instance_id>;
-- Terminate an instance. After you run the kill command, the status of the instance changes to Canceled.
```

Resource quota

A per-process limit on the use of system resources. Two types of quotas are involved: storage and computing. The storage quota is the upper limit of the storage resources configured for a project. If the storage usage approaches the storage quota, an alert is triggered. The computing quota limits the use of memory and CPU resources. The memory and CPUs for the processes that are run in a project at the same time cannot exceed the computing quota.

ACID semantics

This section describes the atomicity, consistency, isolation, durability (ACID) semantics of MaxCompute parallel write jobs.

Terms

- Operation: a single job submitted in MaxCompute.
- Data object: an object that stores data, such as a non-partitioned table or a partition.
- INTO jobs: an SQL job that contains the INTO keyword, such as INSERT INTO and DYNAMIC INSERT INTO.
- OVERWRITE jobs: an SQL job that contains the OVERWRITE keyword, such as INSERT OVERWRITE and DYNAMIC INSERT OVERWRITE.
- Data upload by using Tunnel: an INTO or OVERWRITE job.

Description of ACID semantics

- Atomicity: An operation is completely performed or not performed at all. An operation is not partially performed.
- Consistency: The integrity of data objects is maintained when an operation is performed.
- Isolation: An operation is independent of other parallel operations.
- Durability: After an operation is complete, modified data is permanently valid and not lost even if a system failure occurs.

Description of ACID semantics in MaxCompute

- Atomicity
 - When multiple jobs conflict with each other, MaxCompute ensures that only one job succeeds.
 - The atomicity of the CREATE, OVERWRITE, and DROP operations on a single table or partition is ensured.
 - The atomicity of cross-table operations, such as MULTI-INSERT, cannot be ensured.
 - In extreme cases, the following operations may not be atomic:
 - The DYNAMIC INSERT OVERWRITE operation that is performed on more than 10,000 partitions.
 - An INTO operation. The atomicity of INTO operations cannot be ensured because data cleansing fails during a transaction rollback. However, the data cleansing failure does not cause the loss of original data.
- Consistency
 - The consistency can be ensured for OVERWRITE jobs.
 - If an INTO job fails due to a conflict, data from the failed job may remain.
- Isolation
 - For non-INTO operations, MaxCompute ensures that read operations are submitted.
 - For INTO operations, some read operations may not be submitted.
- Durability

MaxCompute ensures data durability.

5.2. Common commands

5.2.1. Overview

MaxCompute allows you to perform operations on objects, such as projects, tables, resources, and instances. You can run command and execute statements on the client or use SDKs to perform operations on these objects.

This topic describes how to run the commands and execute the statements on the MaxCompute client.

Note

- For more information about how to download and configure the client, see [Configure the client](#).
- For more information about the SDKs, see [SDK for Java](#).

5.2.2. Project operations

This topic describes common statements for project operations.

Create or delete a project

MaxCompute does not provide statements for you to create or delete a project. You can create or delete a project in the Apsara Uni-manager Management Console.

- For more information about how to create a project, see [Create a project](#).
- If you want to delete a project, you can log on to the Apsara Uni-manager Management Console and perform the following steps: Choose **Products** > **Big Data** > MaxCompute. In the left-side navigation pane, click **Projects**. Find the project that you want to delete and click **Delete** in the Actions column.

Go to a project

Syntax:

```
use <project_name>;
```

Description: This statement is used to go to a specified project. After you go to the project, you can directly manage all objects in the project.

Note If the specified project does not exist or you are not added to the project, the system returns an error and exits.

Example:

```
use my_project;
-- my_project is the project that you have permissions to access.
```

Note

- The preceding statement is executed in the MaxCompute client.
- All the statement keywords, project names, table names, and column names in MaxCompute are not case-sensitive.

If the test_src table exists in the my_project project, execute the following statement:

```
select * from test_src;
```

MaxCompute automatically searches for the tables from the my_project project. If the table exists, its data is returned. Otherwise, the system returns an error and exits.

If you are in the my_project project and want to access the test_src table in the my_project2 project, you must specify the project name. Execute the following statement to access the test_src table in the my_project2 project:

```
select * from my_project2.test_src;
```

Data of the my_project2 project, instead of data of the test_src table in the my_project project, is returned.

View projects

Syntax:

```
list projects;
```

Description: This statement is used to query all projects in MaxCompute.

Clear a project

Syntax of the statement that is used to view objects in the project recycle bin:

```
show recyclebin;
```

Description: This statement is used to query all objects in the project recycle bin.

Note Only the project owner is allowed to execute this statement.

Syntax of the statement that is used to clear the project recycle bin:

```
purge all;
```

Description: This statement is used to clear the project recycle bin to release storage space.

 **Note** Only the project owner is allowed to execute this statement.

Syntax of the statement that is used to clear a table:

```
purge table tblname;
```

Description: This statement is used to clear a specified table from the recycle bin to release storage space.

 **Note**

- If the specified table exists, the project owner and users who have write permissions on the table are allowed to execute this statement.
- If the table has been deleted, only the project owner is allowed to execute this statement.

5.2.3. Table operations

This topic describes common statements for table operations.

CREATE TABLE

Syntax:

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [DEFAULT value] [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[LIFECYCLE days];
CREATE TABLE [IF NOT EXISTS] table_name
[AS select_statement | LIKE existing_table_name];
```

Description: This statement is used to create a table.

Examples:

```
CREATE TABLE IF NOT EXISTS sale_detail( shop_name STRING,
customer_id STRING, total_price DOUBLE)
PARTITIONED BY (sale_date STRING, region STRING);
-- Create a partitioned table named sale_detail if no table with this name exists.
```

 **Note**

- Table and column names are not case-sensitive.
- A table or column name can contain only letters, digits, and underscores (_) and must start with a letter. It can be up to 128 bytes in length. Otherwise, an error is returned.
- A comment must be a valid string and can be up to 1,024 bytes in length. Otherwise, an error is returned.
- For more information about the statement, see [Create a table](#).

CHANGEOWNER

Syntax:

```
ALTER TABLE table_name CHANGEOWNER to new_owner;
```

Description: This statement is used to change the owner of a table.

Examples:

```
ALTER TABLE test1 CHANGEOWNER to 'ALIYUN$xxx@aliyun.com';  
-- Change the owner of the test1 table to ALIYUN$xxx@aliyun.com.
```

DROP TABLE

Syntax:

```
DROP TABLE [IF EXISTS] table_name;
```

Description: This statement is used to delete a table.

 **Note** If you do not specify IF EXISTS and the table does not exist, an error is returned. If you specify IF EXISTS, a success message is returned, regardless of whether the table exists.

Parameters:

table_name: the name of the table you want to delete.

Examples:

```
DROP TABLE sale_detail;  
-- If the table exists, a success message is returned.  
DROP TABLE IF EXISTS sale_detail;  
-- A success message is returned regardless of whether the sale_detail table exists.
```

Query table information

Syntax:

```
DESC <table_name>;  
-- Query information about a table or view.  
DESC extended <table_name>;  
-- Query information about an external table.
```

Description: The statements are used to query information about specific tables. Output fields include Owner, Project, CreateTime, LastDDLTime, LastModifiedTime, InternalTable, Size, Native Columns, Partition Columns, and Extended Info. The Owner field indicates the owner of the table. The Project field indicates the project to which the table belongs. The CreateTime field indicates the time when the table was created. The LastDDLTime field indicates the last time when a DDL statement is executed for the table. The LastModifiedTime field indicates the last time when the table data was modified. The InternalTable field indicates that the queried object is a table. The value of this field is always YES. The Size field indicates the storage occupied by table data, in bytes. The Native Columns field shows the names, data types, and comments of standard columns. The Partition Columns field shows the names, data types, and comments of partition key columns. The Extended Info field shows information about an external table, including the storage handler and location.

Parameters:

table_name: In the first statement, this parameter specifies the name of the table or view. In the second statement, this parameter specifies the name of the external table.

Examples:

Execute the following statement to query information about a partitioned table:

```
DESC sale_detail;
```

The following result is returned:

```
+-----+
| Owner: ALIYUN$odpsuser@aliyun.com | Project: test_project |
| TableComment: |
+-----+
| CreateTime:          2014-01-01 17:32:13 |
| LastDDLTime:         2014-01-01 17:57:38 |
| LastModifiedTime:    1970-01-01 08:00:00 |
+-----+
| InternalTable: YES   | Size: 0 |
+-----+
| Native Columns: |
+-----+
| Field          | Type      | Comment |
+-----+
| shop_name     | string    |         |
| customer_id   | string    |         |
| total_price    | double    |         |
+-----+
| Partition Columns: |
+-----+
| sale_date     | string    |         |
| region        | string    |         |
+-----+
```

Note

- If the queried object is a non-partitioned table, the Partition Columns field is not displayed.
- If the queried object is a view, the InternalTable field is replaced with VirtualView whose value is always YES, and the Size field is replaced with ViewText. The ViewText field defines View, such as `select * from src`. For more information about views, see [Create a view](#).

Query partition information

Syntax:

```
desc table_name partition (pt_spec);
```

Description: This statement is used to query information about a partition.

Examples:

Execute the following statement to query information about a partition:

```
desc meta.m_security_users partition (ds='20151010');
```

The following result is returned:

```

+-----+
| PartitionSize: 2109112 |
+-----+
| CreateTime:          2015-10-10 08:48:48 |
| LastDDLTime:         2015-10-10 08:48:48 |
| LastModifiedTime:   2015-10-11 01:33:35 |
+-----+
OK

```

SHOW TABLES and SHOW TABLES LIKE

Syntax:

```

SHOW TABLES;
SHOW TABLES like 'chart';

```

Description:

- SHOW TABLES: queries all tables in the current project.
- SHOW TABLES like 'chart': queries the tables whose names match chart in the current project. You can use regular expressions in the statement to filter tables.

Examples:

Execute the following statement to query all tables in the current project:

```
show tables;
```

Execute the following statement to query the tables whose names match ods_brand:

```
show tables like 'ods_brand*';
```

The following result is returned:

```

ALIYUN$odps_user@aliyun.com:ods_brand
.....

```

Note

- odps_user@aliyun.com: the name of the user who creates the table.
- table_name: the name of the table.

SHOW PARTITIONS

Syntax:

```
SHOW PARTITIONS <table_name>;
```

Description: This statement is used to query all partitions of a table.

Parameters:

table_name: the name of the table that you want to query. If the table does not exist or is a non-partitioned table, an error is returned.

Examples:

Execute the following statement to query all partitions of a table:

```
SHOW PARTITIONS table_name;
```

The following result is returned:

```
partition_col1=col1_value1/partition_col2=col2_value1  
partition_col1=col1_value2/partition_col2=col2_value2
```

Note

- partition_col1 and partition_col2: the partition key columns of the table.
- col1_value1, col2_value1, col1_value2, and col2_value2: the values in the partition key columns.

5.2.4. Instance operations

This topic describes common statements for instance operations.

Show Instances and Show P

Syntax:

```
SHOW INSTANCES [FROM startdate TO enddate] [number];  
SHOW P [FROM startdate TO enddate] [number];  
SHOW INSTANCES [-all];  
SHOW P [-all];  
SHOW P -p <project name>;
```

Description: These statements are used to query information about instances that you created.

Parameters:

- startdate and enddate: the beginning and end of the time range to query. Information about the instances created within the specified period is returned. The dates must be in the format of yyyy-mm-dd. These parameters are optional. If they are not specified, information about instances that you created in the last three days is returned.
- number: the number of instances to return. Information about the specified number of instances submitted at the time nearest to the current time is returned in chronological order. If this parameter is not specified, information about all instances that meet the requirements is returned.
- -all: indicates that information about the instances executed in the current project is returned. By default, a maximum of 50 records are returned. You can execute this statement only when you have the LIST permission on the current project. To return more records, you must use the -limit number option. For example, you can use `show p -all -limit 100` to return information about 100 instances executed in the current project.
- project name: the name of the project. The account that you use must be a member of the project.

Output fields include StartTime, RunTime, Status, InstanceID, and Query. The values of StartTime and RunTime are in seconds. The Query field indicates the SQL statement that corresponds to the instance. The following code provides an example of the output:

Execute the following statement to query the information about instances created by the current user:

```
show p;
```

The following information is returned:

StartTime	RunTime	Status	InstanceID	Owner	Query
2015-04-28 13:57:55	1s	Success	20150428xxxxxxxxxxxxxxxxxxxx	ALIYUN\$xxxxx@aliyun-inner.com	select * from tab_pack_priv limit 20;
...
...

An instance can be in one of the following states:

- **Running:** The instance is running.
- **Success:** Instance operations are complete.
- **Waiting:** The instance is waiting to run.
- **Failed:** The job failed, but the data in the destination table is not modified.
- **Suspended:** The instance is suspended.
- **Cancelled:** The instance is stopped.

Status Instance

Syntax:

```
status instance_id;
```

Description: This statement is used to query the status of a specific instance. Valid values: Success, Failed, Running, and Cancelled.

 **Note** If the instance is not created by the current user, an error is returned.

Parameter: instance_id indicates the ID of the instance whose status you want to query. This parameter uniquely identifies an instance.

Example:

Execute the following statement to query the status of the instance whose ID is 20131225123xxxxxxxxxxxxxxxx.

```
status 20131225123xxxxxxxxxxxxxxxx;
```

The following information is returned:

```
Success
```

Top Instance

Syntax:

```
top instance;  
top instance -all;
```

 **Note** Only project owners or administrators can execute the statement.

Description:

- **top instance:** queries information about running jobs that are submitted by the current user in the current project. Output fields include InstanceID, Owner, Type, StartTime, Progress, Status, Priority, RuntimeUsage (CPU/MEM), TotalUsage (CPU/MEM), and QueueingInfo (POS/LEN).
- **top instance -all:** queries information about all running jobs in the current project. By default, a maximum of 50

records are returned. To return more records, use the `-limit` number option.

Kill Instance

Syntax:

```
kill <instance_id>;
```

Description: This statement is used to stop an instance. You can stop an instance only in the Running state. Note that this is an abnormal process. The execution result of this statement only means that the system has received the request. It does not mean that the job has been stopped. Therefore, you must use `status` to query the instance status.

Parameter: `instance_id` indicates the ID of the instance. This parameter uniquely identifies an instance. It must be the ID of a running instance. Otherwise, an error is returned.

Example:

```
kill 20131225123xxxxxxxxxxxxxxxxxx;
-- Stop the instance whose ID is 20131225123xxxxxxxxxxxxxxxxxx.
```

Desc Instance

Syntax:

```
desc instance <instance_id>;
```

Description: This statement is used to query job information based on the instance ID. Output fields include Query, Owner, Startime, Endtime, and Status.

Parameter: `instance_id` indicates the ID of the instance. This parameter uniquely identifies an instance.

Example:

-- Execute the following statement to query information about the job that corresponds to the instance whose ID is 20150715xxxxxxxxxxxxxxxxxx.

```
desc instance 20150715xxxxxxxxxxxxxxxxxx;
```

The following information is returned:

```
ID                20150715xxxxxxxxxxxxxxxxxx
Owner             ALIYUN$XXXXXX@alibaba-inc.com
StartTime         2015-07-15 18:34:41
EndTime          2015-07-15 18:34:42
Status            Terminated
console_select_query_task_1436956481295 Success
Query             select * from mj_test;
```

Wait Instance

Syntax:

```
wait instance_id;
```

Description: This statement is used to obtain the operational logs of the job based on the instance ID. The returned information includes a Logview URL. You can log on to Logview to view log details.

 **Note** You can obtain the Logview URL of an instance created for more than three days. However, you cannot use the URL to go to Logview because it expires and is cleared.

Parameter: `instance_id` indicates the ID of the instance. This parameter uniquely identifies an instance.

Sample statements:

Execute the following statement to query the operational logs of the job that corresponds to the instance whose ID is 20170925161122379gxxxxxx.

```
wait 20170925161122379gxxxxxx;
```

The following information is returned:

```
ID = 20170925161122379g3xxxxxx
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=aliam&i=20170925161122379g3xxxxxx&token=XXXXXXXXvbiI6IjEifQ==
Job Queueing...
Summary:
resource cost: cpu 0.05 Core * Min, memory 0.05 GB * Min
inputs:
    alian.bank_data: 41187 (588232 bytes)
outputs:
    alian.result_table: 8 (640 bytes)
Job run time: 2.000
Job run mode: service job
Job run engine: execution engine
M1:
    instance count: 1
    run time: 1.000
    instance time:
        min: 1.000, max: 1.000, avg: 1.000
    input records:
        TableScan_REL5213301: 41187 (min: 41187, max: 41187, avg: 41187)
)
    output records:
        StreamLineWrite_REL5213305: 8 (min: 8, max: 8, avg: 8)
R2_1:
    instance count: 1
    run time: 2.000
    instance time:
        min: 2.000, max: 2.000, avg: 2.000
    input records:
        StreamLineRead_REL5213306: 8 (min: 8, max: 8, avg: 8)
    output records:
        TableSink_REL5213309: 8 (min: 8, max: 8, avg: 8)
```

5.2.5. Resource operations

This topic describes common statements for resource operations.

Add a resource

Syntax:

```
add file <local_file> [as alias] [comment 'cmt'][-f];
add archive <local_file> [as alias] [comment 'cmt'][-f];
add table <table_name> [partition (spec)] [as alias] [comment 'cmt'][-f];
add jar <local_file.jar> [comment 'cmt'][-f];
add py <local_file.py> [comment 'cmt'][-f];
```

Description: The statements are used to add resources to MaxCompute.

Parameter	Description
file/archive/table/jar/py	The type of the resource.
local_file	The path of the file that you want to add. The file name is used as the resource name. A resource name is a unique identifier of a resource.
table_name	The name of the MaxCompute table.
[partition (spec)]	If the resource that you want to add is a partitioned table, MaxCompute takes only a partition, not the whole partitioned table, as a resource.
alias	The name of the resource. If this parameter is not specified, the file name is used as the resource name. This feature does support JAR and PY resources.
[comment 'cmt']	The comment of the resource.
[-f]	If a resource with the same name exists, this operation overwrites the existing resource. If this parameter is not specified and a resource with the same name exists, the operation fails.

Examples:

-- Add a table resource whose alias is sale.res to MaxCompute.

```
add table sale_detail partition (ds='20170602') as sale.res comment 'sale detail on 201706 02' -f;
```

The following result is returned:

```
OK: Resource 'sale.res' have been updated.
```

 **Note** The size of each resource file cannot exceed 500 MB. The total size of resource files referenced by a single SQL or MapReduce job cannot exceed 2,048 MB.

Delete a resource

Syntax:

```
DROP RESOURCE <resource_name>;
```

Description: This statement is used to delete an existing resource.

Parameters:

resource_name: the name of the resource that you want to delete.

Examples:

Execute the following statement to delete an existing resource:

```
DROP RESOURCE getaddr.jar;
```

The following result is returned:

```
Confirm to "DROP RESOURCE getaddr.jar" (yes/no)? y
-- Enter y to confirm your input.
OK
```

Query resources

Syntax:

```
LIST RESOURCES;
```

Description: This statement is used to query all resources in the current project.

Examples:

Execute the following statement to query all resources in the current project:

```
list resources;
```

The following result is returned:

Resource Name	Comment	Last Modified Time	Type
1234.txt		2014-02-27 07:07:56	file
mapred.jar		2014-02-27 07:07:57	jar

Download a resource

Syntax:

```
GET RESOURCE <resource_name> <path>;
```

Description: This statement is used to download a resource from MaxCompute to your computer. You can download file, JAR, archive, and PY resources, but not table resources.

Parameters:

- `resource_name`: the name of the resource that you want to download.
- `path`: the path to save the resource.

Examples:

Execute the following statement to download a resource from MaxCompute to your computer:

```
get resource odps-udf-examples.jar d:\;
```

The following result is returned:

```
OK
```

5.2.6. Function operations

This topic describes common statements for function operations.

Create a function

Syntax:

```
CREATE FUNCTION <function_name> AS <package_to_class> USING<resource_list>;
```

The following table describes the parameters in this statement.

Parameter	Description
function_name	The name of the user-defined function (UDF). This name is used in SQL statements to reference this function.
package_to_class	For a Java UDF, package_to_class indicates a fully qualified class name. It consists of names from the top-level package name all the way to the class name for UDF implementation. For a Python UDF, package_to_class indicates the Python script name and class name. The value must be enclosed in a pair of single quotation marks ('').
resource_list	The list of resources that the UDF uses. The list must include the resource where the UDF code is located. Make sure that the resources are uploaded to MaxCompute before you register the function. If the user code reads resource files by using the distributed cache API, this list must also include all resource files that are read by the UDF. If the resource list contains multiple resources, separate them with commas (,). The resource list must be enclosed in a pair of single quotation marks (''). If you need to specify the project where the resource is located, use the <project_name>/resources/<resource_name> format.

Examples:

- Assume that Java UDF class org.alidata.odps.udf.examples.Lower is in my_lower.jar. Execute the following statement to create the my_lower function:

```
CREATE FUNCTION test_lower AS 'org.alidata.odps.udf.examples.Lower'
USING 'my_lower.jar';
```

- Assume that Python UDF class MyLower is in the pyudf_test.py script of the test_project project. Execute the following statement to create the my_lower function:

```
create function my_lower as 'pyudf_test.MyLower'
using 'pyudf_test.py';
```

- Assume that Java UDF class com.aliyun.odps.examples.udf.UDTFResource is in udtfexample1.jar and the function depends on file resource file_resource.txt, table resource table_resource1, and archive resource test_archive.zip. Execute the following statement to create the test_udtf function:

```
create function test_udtf as 'com.aliyun.odps.examples.udf.UDTFResource'
using 'udtfexample1.jar, file_resource.txt, table_resource1, test_archive.zip';
```

Note

- Similar to resource file names, function names must be unique.
- Only project owners have the permissions to use UDFs to overwrite built-in functions. If you use a UDF that overwrites a built-in function, warning information is included in the Summary parameter after the SQL statement is executed.

Delete a function

Syntax:

```
DROP FUNCTION <function_name>;
```

Parameters:

function_name: the name of the function you want to delete.

Examples:

```
DROP FUNCTION test_lower;
```

Query functions

Syntax:

```
LIST FUNCTIONS;  
-- Query all UDFs in the current project.  
LIST FUNCTIONS -p project_name;  
-- Query all UDFs in a specific project.
```

5.2.7. Time zone configuration operations

This topic describes how to use a SET command to configure the time zone for a MaxCompute project.

Time zone configurations

The default time zone of a MaxCompute project is UTC+8. Time-related built-in functions and the fields of the DATETIME, TIMESTAMP, and DATE types are calculated based on UTC+8. You can use one of the following methods to configure the time zone:

- Session level: Run the `set odps.sql.timezone=<timezoneid>;` command along with a computing statement.

Examples:

Run the following command to set the time zone to Asia/Tokyo:

```
set odps.sql.timezone=Asia/Tokyo;
```

Run the following command to query the current time zone:

```
select getdate();
```

The following result is returned:

```
+-----+  
| _c0      |  
+-----+  
| 2018-10-30 23:49:50 |  
+-----+
```

- Project level: Run the `setProject odps.sql.timezone=<timezoneid>;` command as the project owner.

 **Notice** After the time zone of a project is configured, it is used for all time computing, and the data of existing jobs is affected. Therefore, exercise caution when you perform this operation. We recommend that you set time zones only for new projects.

Limits and usage notes

- SQL built-in date functions, user-defined functions (UDFs), user-defined types (UDTs), user-defined joins (UDJs), and SELECT TRANSFORM allow you to obtain the timezone attribute of a project to configure the time zone.
- A time zone must be configured in the format such as Asia/Shanghai, which supports daylight saving time. Do not configure it in the GMT+9 format.
- If the time zone of the SDK differs from that of the project, you must configure the GMT time zone to convert the data type from DATETIME into STRING.
- After you configure the time zone, the output time may be different from the real time after you run the related SQL statements by using DataWorks. For data that is between the years 1900 and 1928, the time difference is 352 seconds. For data that is before the year 1900, the time difference is 9 seconds.
- MaxCompute, SDK for Java, and the related client are updated to ensure that DATETIME data stored in MaxCompute is accurate across multiple time zones. The updated versions of the client and SDK have the -oversea suffix. The update may affect the display of DATETIME data earlier than January 1, 1928 in MaxCompute.
- If the time zone is not UTC+8, we recommend that you update SDK for Java and the client. This ensures that the SQL-based computing results and data transferred by using Tunnel after January 1, 1900 are accurate and consistent. After the update, for the DATETIME data before January 1, 1900, the SQL-based computing results and data transferred by using Tunnel may still have a difference of 343 seconds. For the DATETIME data between January 1, 1900 and January 1, 1928 uploaded before the update, the time in the new version is 352 seconds earlier.
- If you do not update SDK for Java and the client to the version with the -oversea suffix, the SQL-based computing results and data transferred by using Tunnel still have differences. For data before January 1, 1900, the time difference is 9 seconds. For data between January 1, 1900 and January 1, 1928, the time difference is 352 seconds.

 **Note** The modification of time zone configurations for the new client or SDK for Java does not affect the time zone configuration in DataWorks. Therefore, the time zones may vary. You must evaluate the impact on scheduled tasks in DataWorks.

- If you use a third-party client that is connected to MaxCompute by using JDBC, you must configure the time zone on the client to ensure time consistency between the client and the server.
- MaxCompute MapReduce, Machine Learning Platform for AI, and MaxCompute Graph support the time zone configuration.
- MaxCompute Spark supports the time zone configuration.
 - If tasks are submitted to a MaxCompute computing cluster, the time zone of the project can be automatically obtained.
 - If tasks are submitted from spark-shell, spark-sql, or pyspark in yarn-client mode, you must configure the Spark-defaults.conf parameter of the driver and add spark.driver.extraJavaOptions - Duser.timezone=America/Los_Angeles. The timezone parameter indicates the time zone that you want to use.

5.2.8. Tunnel operations

This topic describes common commands for Tunnel operations.

Use of Tunnel commands

You can run the `tunnel help;` subcommand on the MaxCompute client to query help information. The following result is returned:

```
Usage: tunnel <subcommand> [options] [args]
Type 'tunnel help <subcommand>' for help on a specific subcommand.
Available subcommands:
  upload (u)
  download (d)
  resume (r)
  show (s)
  purge (p)
  help (h)
tunnel is a command for uploading data to / downloading data from ODPS.
```

Parameters:

- **upload:** uploads data to a MaxCompute table. You can upload files or level-1 directories to only one table or partition each time. For a partitioned table, you must specify the partition to which you want to upload data. For a multi-level partitioned table, you must specify the last-level partition.

```
tunnel upload log.txt test_project.test_table/p1="b1",p2="b2";
-- Upload data in the log.txt file to the p1="b1" and p2="b2" partitions of the test_table table that has two levels of partitions in the test_project project.
tunnel upload log.txt test_table --scan=true;
-- Upload data in the log.txt file to the test_table table. The scan parameter is used to check whether data in the log.txt file complies with the schema of the test_table table. If it does not, the system reports an error and stops the upload.
```

- **download:** downloads data from a MaxCompute table. You can download data from only one table or partition to a file each time. For a partitioned table, you must specify the partition from which you want to download data. If the table has multiple levels of partitions, you must specify the last-level partition.

```
tunnel download test_project.test_table/p1="b1",p2="b2" test_table.txt;
-- Download data from the test_project.test_table table that has two levels of partitions to the test_table.txt file.
```

- **resume:** resumes the transfer of files or directories. The transfer is interrupted because your network is disconnected or Tunnel is faulty. You can use this command to resume only data uploads. One data download or upload is referred to as a session. You must specify the session ID in the RESUME command before you run this command.

```
tunnel resume;
```

- **show:** shows historical task information.

```
tunnel show history -n 5;
-- Show the commands used in the last five data uploads or downloads.
tunnel show log;
-- Show the logs of the last data upload or download.
```

- **purge:** clears a session directory. By default, sessions from the last three days are cleared.

```
tunnel purge 5;
--Clear logs from the last five days.
```

- **help:** obtains help information.

Upload

Description: This command is used to upload data to a MaxCompute table in append mode.

You can run the `tunnel help upload;` subcommand to query help information. The following result is returned:

```
usage: tunnel upload [options] <path> <[project.]table[/partition]>
       upload data from local file
-acp,-auto-create-partition <ARG>    auto create target partition if not
                                       exists, default false
-bs,-block-size <ARG>                block size in MiB, default 100
-c,-charset <ARG>                    specify file charset, default ignore.
                                       set ignore to download raw data
-cf,-csv-format <ARG>                use csv format (true|false), default
                                       false. When uploading in csv format,
                                       file splitting not supported.
-cp,-compress <ARG>                  compress, default true
-dbr,-discard-bad-records <ARG>      specify discard bad records
                                       action(true|false), default false
-dfp,-date-format-pattern <ARG>      specify date format pattern, default
                                       yyyy-MM-dd HH:mm:ss
-fd,-field-delimiter <ARG>           specify field delimiter, support
                                       unicode, eg \u0001. default ",",
-h,-header <ARG>                     if local file should have table
                                       header, default false
-mbr,-max-bad-records <ARG>          max bad records, default 1000
-ni,-null-indicator <ARG>            specify null indicator string,
                                       default "" (empty string)
-ow,-overwrite <true | false>        overwrite specified table or
                                       partition, default: false
-rd,-record-delimiter <ARG>          specify record delimiter, support
                                       unicode, eg \u0001. default "\r\n"
-s,-scan <ARG>                        specify scan file
                                       action(true|false|only), default true
-sd,-session-dir <ARG>               set session dir, default
                                       D:\software\odpscmd_public\plugins\dship
-ss,-strict-schema <ARG>             specify strict schema mode. If false,
                                       extra data will be abandoned and
                                       insufficient field will be filled
                                       with null. Default true
-t,-threads <ARG>                    number of threads, default 1
-te,-tunnel_endpoint <ARG>           tunnel endpoint
-time,-time <ARG>                    keep track of upload/download elapsed
                                       time or not. Default false
-tz,-time-zone <ARG>                 time zone, default local timezone:
                                       Asia/Shanghai
```

Example:

```
tunnel upload log.txt test_project.test_table/p1="b1",p2="b2"
```

The following table describes the parameters in this statement.

Parameter	Description
path	The path and name of the file that you want to upload.
[project.]table[/partition]	The name of the table to which you want to upload data. You must specify the last-level partition for a partitioned table. If the table does not belong to the current project, you must specify the project where the table is located.
-acp	The partition to which you want to upload data. If the specified partition does not exist, a partition is automatically created. Default value: False.

Parameter	Description
-bs	The size of the data block uploaded by Tunnel each time. Default value: 100 MiB. 1 MiB = 1024 × 1024 bytes.
-c	The encoding format of the data file. The default value is UTF-8 without timing. By default, the source data is downloaded.
-cf	Specifies whether the file is a CSV file. Default value: False.
-cp	Specifies whether to compress the file before you upload it to MaxCompute to reduce network traffic. Default value: True.
-dbr	Specifies whether to omit dirty data, such as additional columns, missing columns, or unmatched types of column data. The value True indicates that all data that does not match the schema of the table is omitted. The value False indicates that an error is returned when dirty data is detected. This ensures that raw data in the table to which you want to upload data is not contaminated. Default value: False.
-dfp	The format of DATETIME data. The default format is yyyy-MM-dd HH:mm:ss. If you want to specify the DATETIME data that is accurate to the millisecond, the format <code>tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'</code> can be used.
-fd	The column delimiter for the data file. The default value is a comma (,).
-h	Specifies whether the data file has table headers. If this parameter is set to True, the data from the second row in the file is uploaded. Default value: False.
-mbr	The maximum number of dirty data records. If the number of dirty data records exceeds the value of this parameter, the upload stops. Default value: 1000.
-ni	The NULL data identifier. The default value is an empty string.
-ow	Specifies whether the uploaded data overwrites the table or partition. Default value: False, which indicates that data is appended.
-rd	The row delimiter for the data file. The default value is \n in a Linux operating system or \r\n in a Windows operating system.
-s	Specifies whether to scan the data file. Default value: True. The value True indicates that the system scans the data and starts to import it only if the data is in the correct format. The value False indicates that the system imports data without scanning. The value Only indicates that the system only scans the data but does not import it.
-sd	The path of the session directory. Default value: <code>D:/console/plugins/dship/lib/...</code>
-ss	The strict schema mode. Default value: True. If the parameter is set to False, extra data is discarded and the fields that are not specified are filled with NULL.
-t	The number of threads. Default value: 1.
-te	The endpoint of Tunnel.

Parameter	Description
-time	Specifies whether the upload time is tracked. Default value: False.
-tz	The time zone. By default, the local time zone is used. Example: Asia/Shanghai.

Example:

1. Execute the following statement to create a partitioned table named `sale_detail`:

```
CREATE TABLE IF NOT EXISTS sale_detail(
  shop_name STRING,
  customer_id STRING,
  total_price DOUBLE)
PARTITIONED BY (sale_date STRING, region STRING);
```

2. Execute the following statement to add a partition to the `sale_detail` table:

```
alter table sale_detail add partition (sale_date='201705', region='hangzhou');
```

3. Prepare the `data.txt` file that contains the following data and save the file in the `d:\data.txt` directory:

```
shop9,97,100
shop10,10,200
shop11,11
```

Note In this example, the third row of the `data.txt` file does not comply with the schema of the `sale_detail` table. The `sale_detail` table defines three columns, but the third row of this file contains only two columns.

4. Run the following command to upload the `data.txt` file to the `sale_detail` table.

```
tunnel u d:\data.txt sale_detail/sale_date=201705,region=hangzhou -s false;
```

The data upload fails because the `data.txt` file contains dirty data. The system returns the session ID and an error message.

```
Upload session: 201706101639224880870a002ec60c
Start upload:d:\data.txt
Total bytes:41 Split input to 1 blocks
2017-06-10 16:39:22 upload block: '1'
ERROR: column mismatch -,expected 3 columns, 2 columns found, please check data or delimiter
```

5. Execute the following statement to verify the data:

```
select * from sale_detail where sale_date='201312';
```

The data upload fails because dirty data is detected. As a result, the `sale_detail` table is empty.

```
ID = 20150610xxxxxxxxxxxxvc61z5
+-----+-----+-----+-----+-----+
| shop_name | customer_id | total_price | sale_date | region |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

6. Run the following command to query the ID of the failed session.

```
tunnel show history;
```

The following result is returned:

```
20150610xxxxxxxxxx70a002ec60c failed 'u --config-file /D:/console/conf/odps_config.ini --project odpstest_ay52c_ay52 --endpoint http://service.cn-shanghai.maxcompute.aliyun.com/api --id UlxxxxxxxxxxrI1 --key 2m4r3WvTxxxxxxxxx0InVke7UkvR d:\data.txt sale_detail/sale_date=201312,region=hangzhou -s false'
```

7. Modify the data.txt file to comply with the schema of the sale_detail table.

```
shopx,x_id,100
shopy,y_id,200
```

8. Run the RESUME command to resume the data upload. In the command, 20150610xxxxxxxxxx70a002ec60c is the ID of the failed session.

```
tunnel resume 20150610xxxxxxxxxx70a002ec60c --force;
```

The following information is returned:

```
start resume
20150610xxxxxxxxxx70a002ec60c
Upload session: 20150610xxxxxxxxxx70a002ec60c
Start upload:d:\data.txt
Resume 1 blocks
2015-06-10 16:46:42 upload block: '1'
2015-06-10 16:46:42 upload block complete, blockid=1
upload complete, average speed is 0 KB/s
OK
```

9. Execute the following statement to verify the data:

```
select * from sale_detail where sale_date='201312';
```

If the following result is returned, the upload succeeds.

```
ID = 20150610xxxxxxxxxxa741z5
+-----+-----+-----+-----+-----+
| shop_name | customer_id | total_price | sale_date | region |
+-----+-----+-----+-----+-----+
| shopx     | x_id        | 100.0       | 201312   | hangzhou|
| shopy     | y_id        | 200.0       | 201312   | hangzhou|
+-----+-----+-----+-----+-----+
```

Download

Description: This command is used to download MaxCompute table data or the execution result of a specific instance to your computer.

You can run the `tunnel help download;` subcommand to query help information. The following result is returned:

```
Usage: tunnel download [options] <[project.]table[/partition]> <path>
download data to local file
-c,-charset <ARG>          specify file charset, default ignore.
                           set ignore to download raw data
-cf,-csv-format <ARG>     use csv format (true|false), default
                           false. When uploading in csv format,
                           file splitting not supported.
-ci,-columns-index <ARG>  specify the columns index(starts from
                           0) to download, use comma to split each
                           index
```

```

-cn,-columns-name <ARG>      specify the columns name to download,
                             use comma to split each name
-cp,-compress <ARG>         compress, default true
-dfp,-date-format-pattern <ARG> specify date format pattern, default
                             yyyy-MM-dd HH:mm:ss
-e,-exponential <ARG>      When download double values, use
                             exponential express if necessary.
                             Otherwise at most 20 digits will be
                             reserved. Default false
-fd,-field-delimiter <ARG>  specify field delimiter, support
                             unicode, eg \u0001. default ",",
-h,-header <ARG>           if local file should have table header,
                             default false
    -limit <ARG>           specify the number of records to
                             download
-ni,-null-indicator <ARG>  specify null indicator string, default
                             ""(empty string)
-rd,-record-delimiter <ARG> specify record delimiter, support
                             unicode, eg \u0001. default "\r\n"
-sd,-session-dir <ARG>    set session dir, default
                             D:\software\odpscmd_public\plugins\dship
-t,-threads <ARG>         number of threads, default 1
-te,-tunnel_endpoint <ARG> tunnel endpoint
-time,-time <ARG>         keep track of upload/download elapsed
                             time or not. Default false
-tz,-time-zone <ARG>     time zone, default local timezone:
                             Asia/Shanghai
usage: tunnel download [options] instance://<[project/]instance_id> <path>
       download instance result to local file
-c,-charset <ARG>         specify file charset, default ignore.
                             set ignore to download raw data
-cf,-csv-format <ARG>    use csv format (true|false), default
                             false. When uploading in csv format,
                             file splitting not supported.
-ci,-columns-index <ARG> specify the columns index(starts from
                             0) to download, use comma to split each
                             index
-cn,-columns-name <ARG>  specify the columns name to download,
                             use comma to split each name
-cp,-compress <ARG>     compress, default true
-dfp,-date-format-pattern <ARG> specify date format pattern, default
                             yyyy-MM-dd HH:mm:ss
-e,-exponential <ARG>  When download double values, use
                             exponential express if necessary.
                             Otherwise at most 20 digits will be
                             reserved. Default false
-fd,-field-delimiter <ARG> specify field delimiter, support
                             unicode, eg \u0001. default ",",
-h,-header <ARG>       if local file should have table header,
                             default false
    -limit <ARG>       specify the number of records to
                             download
-ni,-null-indicator <ARG> specify null indicator string, default
                             ""(empty string)
-rd,-record-delimiter <ARG> specify record delimiter, support
                             unicode, eg \u0001. default "\r\n"
-sd,-session-dir <ARG>  set session dir, default
                             D:\software\odpscmd_public\plugins\dshi
-t,-threads <ARG>     number of threads, default 1

```

```

-te, -tunnel_endpoint <ARG>      tunnel endpoint
-time, -time <ARG>                keep track of upload/download elapsed
                                  time or not. Default false
-tz, -time-zone <ARG>            time zone, default local timezone:
                                  Asia/Shanghai

Example:
  tunnel download test_project.test_table/p1="b1",p2="b2" log.txt // Download data from a specific
  table.
  tunnel download instance://test_project/test_instance log.txt // Download the execution result
  of a specific instance.
    
```

The following table describes the parameters in this command.

Parameter	Description
path	The path in which the downloaded data file is saved.
[project.]table[/partition]	The name of the table that you want to download. You must specify the last-level partition for a partitioned table. If the table does not belong to the current project, you must specify the project where the table is located.
[project/]instance_id	The ID of the instance. You must specify this parameter to download the execution result of a specific instance.
-fd	The column delimiter for the data file. The default value is a comma (,).
-rd	The row delimiter for the data file. Default value: \r\n.
-dfp	The format of DATETIME data. The default format is yyyy-MM-dd HH:mm:ss.
-ni	The NULL data identifier. The default value is an empty string.
-c	The encoding format of the data file. Default value: UTF-8.
-cf	Specifies whether the file is a CSV file. Default value: False.
-ci	Specifies that column indexes (starting from 0) are downloaded. Separate column indexes with commas (,).
-cp	The name of the column to download. Separate multiple column names with commas (,).
-e	Specifies whether the data of the DOUBLE type you want to download is represented by an exponential function if required. If the data is not represented by an exponential function, a maximum of 20 bits are retained. Default value: False.
-h	Specifies whether the data file has table headers. Default value: False. If this parameter is set to True, the data from the second row in the file is downloaded. <div style="border: 1px solid #add8e6; padding: 5px; background-color: #e6f2ff;"> <p> Note -h=true and threads>1 cannot be used together. threads>1 indicates multiple threads.</p> </div>
-limit	The number of files you want to download.
-tz	The time zone. By default, the local time zone is used. Example: Asia/Shanghai.
-time	Specifies whether the upload time is tracked. Default value: False.

Parameter	Description
-te	The endpoint of Tunnel.
-t	The number of threads. Default value: 1.
-sd	The path of the session directory. Default value: <i>D:/console/plugins/dship/lib/...</i>

Example:

1. Execute the following statement to query the `sale_detail` table:

```
select * from sale_detail;
```

The following information is returned:

```
ID = 20170724071705393ge3csfb8
... ..
```

2. Run the following command to download the execution results to a file on the on-premises machine:

```
tunnel download instance://20170724071705393ge3csfb8 result;
```

If the following result is returned, the download succeeds.

```
2017-07-24 15:18:47 - new session: 2017072415184785b6516400090ca8    total lines: 8
2017-07-24 15:18:47 - file [0]: [0, 8), result
downloading 8 records into 1 file
2017-07-24 15:18:47 - file [0] start
2017-07-24 15:18:48 - file [0] OK. total: 44 bytes
download OK
```

After you enable `use_instance_tunnel` on the MaxCompute client, you can use `InstanceTunnel` to download the query results that are returned by `SELECT` statements. This helps you download query results of any size at any time. You can use one of the following methods to enable this feature:

- On the client, set `use_instance_tunnel` to `true` in the `odps_config.ini` file and make sure that `instance_tunnel_max_record` is automatically set to 10000.

```
# download sql results by instance tunnel
use_instance_tunnel=true
# the max records when download sql results by instance tunnel
instance_tunnel_max_record=10000
```

Note The `instance_tunnel_max_record` parameter specifies the maximum number of SQL result records that can be downloaded by using `InstanceTunnel`. If you do not specify this parameter, the number of result records that can be downloaded is not limited.

- Set `console.sql.result.instancetunnel` to `true`.

```
set console.sql.result.instancetunnel=true;
```

Resume

Description: This command is used to resume historical operations. Only data uploads can be resumed.

You can run the `tunnel help resume;` subcommand to query help information. The following result is returned:

```
Usage: tunnel resume [session_id] [--force]
       resume an upload session
-f,--force force resume
Example:
tunnel resume
```

Parameters:

- session_id: the session ID of the failed data upload. You must specify this parameter.
- -f: specifies whether to forcibly resume historical operations. By default, this parameter is not specified.

Example:

Run the RESUME command to resume the session for which the upload failed. In this command, 20150610xxxxxxxxxx70a002ec60c is the ID of the session for which the upload failed.

```
tunnel resume 20150610xxxxxxxxxx70a002ec60c -force;
```

The following information is returned:

```
start resume
201706101639224880870a002ec60c
Upload session: 201706101639224880870a002ec60c
Start upload:d:\data.txt
Resume 1 blocks
2017-06-10 16:46:42 upload block: '1'
2017-06-10 16:46:42 upload block complete, blockid=1
upload complete, average > speed is 0 KB/s
OK
```

Show

Description: This command is used to show historical records.

You can run the `tunnel help show;` subcommand to query help information. The following result is returned:

```
Usage: tunnel show history [options]
       show session information
-n,-number <ARG> lines
Example:
tunnel show history -n 5
tunnel show log
```

Parameter: -n specifies the number of rows to be displayed.

Purge

Description: This command is used to clear a session directory.

You can run the `tunnel help purge;` subcommand to query help information. The following result is returned:

```
usage: tunnel purge [n]
       force session history to be purged.([n] days before, default
       3 days)
Example:
tunnel purge 5
```

Parameter: n specifies after how many days historical logs are cleared. Default value: 3.

5.2.9. Other operations

This topic describes common statements for other operations.

ALIAS

The ALIAS statement is used to create an alias for a resource. You can create the same alias for different resources to read these resources from the MaxCompute MapReduce or UDF code without code modification.

Syntax:

```
ALIAS <alias>=<real>;
```

Description: This statement is used to create an alias for a resource.

Parameters:

- alias: the alias of the resource.
- real: the original name of the resource.

Sample statements:

```
ADD TABLE src_part PARTITION (ds='20171208') AS res_20171208;
ADD TABLE src_part PARTITION (ds='20171209') AS res_20171209;
-- Add the res_20171208 and res_20171209 resources.
ALIAS resName=res_20171208;
jar -resources resName -libjars work.jar -classpath ./work.jar com.company.MainClass args ...;
-- Set the alias of the res_20171208 resource to resName and call this resource.
ALIAS resName=res_20171209;
jar -resources resName -libjars work.jar -classpath ./work.jar com.company.MainClass args ...;
-- Set the alias of the res_20171209 resource to resName and call this resource.
```

 **Note** In the preceding example, different resource tables are referenced by the same resource alias resName in two jobs. Different data is read with the same code.

Set

Syntax:

```
set <KEY>=<VALUE>
```

Description: This statement is used to configure built-in or user-defined system variables of MaxCompute to affect MaxCompute execution rules.

Parameters:

- KEY: the name of the attribute that you want to set.
- VALUE: the value of the attribute.

MaxCompute SQL and the latest version of MaxCompute MapReduce support the following SET statements:

```

set odps.stage.mapper.mem=
-- Set the memory size of each mapper. Unit: MB. Default value: 1024.
set odps.stage.reducer.mem=
-- Set the memory size of each reducer. Unit: MB. Default value: 1024.
set odps.stage.joiner.mem=
-- Set the memory size of each joiner. Unit: MB. Default value: 1024.
set odps.stage.mem =
-- Set the total memory size of all workers in a specific MaxCompute job. This statement has a lower
priority than the preceding three statements. The value is in MB. No default value is defined.
set odps.stage.mapper.split.size=
-- Set the input data volume of each mapper to indirectly control the number of workers in each map
stage. The input data volume indicates the size of each slice in the input file. Unit: MB. Default v
alue: 256.
set odps.stage.reducer.num=
-- Set the number of workers in each reduce stage. No default value is defined.
set odps.stage.joiner.num=
-- Set the number of workers in each join stage. No default value is defined.
set odps.stage.num=
-- Modify the parallism of workers at all stages in a specific MaxCompute job. This statement has a
lower priority than the preceding three statements. No default value is defined.

```

The early versions of MaxCompute MapReduce support the following SET statements:

```

set odps.mapred.map.memory=
-- Set the memory size of each mapper. Unit: MB. Default value: 1024.
set odps.mapred.reduce.memory=
-- Set the memory size of each reducer. Unit: MB. Default value: 1024.
set odps.mapred.map.split.size=
-- Set the input data volume of each mapper to indirectly control the number of workers in each map
stage. The input data volume indicates the size of each slice in the input file. Unit: MB. Default v
alue: 256.
set odps.mapred.reduce.tasks=
-- Set the number of workers in each reduce stage. No default value is defined.

```

Examples:

- Adjust the cache size pre-defined for a complex column when data is written to MaxCompute tables to improve write performance.

```
set odps.sql.executionengine.coldata.deep.buffer.size.max=1048576;
```

- Set the size of data read by each mapper to 256 MB.

```
set odps.stage.mapper.split.size=256;
```

SetProject

Syntax

```
setproject ["<KEY>=<VALUE>"];
```

Description: This statement is used to configure project attributes. If <KEY>=<VALUE> is not specified, the current attribute configurations of the project are displayed.

The following table describes project attributes.

Attribute	Permission owner	Description	Value Range
odps.table.drop.ignorenonexistent	All users	Indicates whether to report an error when you try to delete a table that does not exist. If the value is True, no error is reported.	True and False
odps.instance.priority.autoadjust	Project owner	Indicates whether to automatically prioritize small tasks.	True and False
odps.instance.priority.level	Project owner	Indicates the priority of a task in a project.	1~3  Note The value 1 indicates the highest priority.
odps.security.ip.whitelist	Project owner	Indicates an IP address whitelist for the project.	List of IP addresses separated by commas (,)
odps.table.lifecycle	Project owner	optional: The LIFECYCLE clause is optional in the statement that is used to create a table. If no lifecycle is set for a table, the table does not expire. mandatory: The LIFECYCLE clause is required. inherit: If no lifecycle is set for a table, the value of odps.table.lifecycle.value is the lifecycle of this table.	optional, mandatory, and inherit
odps.table.lifecycle.value	Project owner	Indicates the default lifecycle.	1~37231  Note The default value is 37231.
odps.instance.remain.days	Project owner	Indicates the retention period for instance information, in days.	3~30
odps.task.sql.outerjoin.ppd	Project owner	Indicates whether the filter conditions in FULL OUTER JOIN are pushed down.	True and False

Attribute	Permission owner	Description	Value Range
odps.function.strictmode	Project owner	Indicates whether to return NULL or report an error when dirty data is involved in the process of using built-in functions. The value False indicates that NULL is returned. The value True indicates that an error is returned.	True and False
odps.task.sql.write.str2null	Project owner	Indicates whether to consider empty strings as NULL. If the value is True, empty strings are considered NULL.	True and False

EXPORT

Syntax:

```
export <projectname> <local_path>;
```

Description: This statement is used to export the metadata of a project to your computer. Metadata is represented by statements acceptable by the MaxCompute client (odpscmd). The exported metadata file can be used to recreate a project.

Show Flags

Syntax:

```
show flags;
```

Description: This statement is used to show the parameters configured by using SET statements.

 **Note** This statement takes effect only at the project level.

COST SQL

Syntax:

```
cost sql <SQL Sentence>;
```

Description: This statement is used to estimate the costs of an SQL statement based on the size of the input data, the number of UDFs, and the SQL complexity.

Example:

Execute the following statement to estimate the costs of an SQL statement:

```
cost sql select distinct project_name, user_name from meta.m_security_users distribute by project_name sort by project_name;
```

The following information is returned:

```
ID = 20190715113033121xxxxxxx  
Input:65727592 Bytes  
UDF:0  
Complexity:1.0
```

6. MaxCompute SQL

6.1. Overview

6.1.1. Scenarios

This topic describes the scenarios of MaxCompute SQL.

MaxCompute SQL offline computing is applicable to scenarios where you need to process terabytes of data but do not require real-time processing. It requires a relatively long time to prepare and submit jobs for MaxCompute SQL offline computing. Therefore, MaxCompute SQL is not well-suited for scenarios where thousands or even tens of thousands of transactions need to be processed per second. MaxCompute SQL online computing provides near real-time (NRT) processing capabilities.

The MaxCompute SQL syntax is similar to the SQL syntax. The MaxCompute SQL syntax can be considered a subset of standard SQL. MaxCompute SQL lacks many database features, such as transactions, primary key constraints, and indexes. Therefore, MaxCompute SQL cannot be regarded as a database. The maximum size of each SQL statement supported by MaxCompute is 2 MB.

6.1.2. Reserved words

The keywords of SQL statements are reserved words in MaxCompute. Do not use the reserved words when you name tables, columns, or partitions. If you use the reserved words, an error is reported. Reserved words are not case-sensitive.

This section lists the common reserved words. For a complete list of reserved words, see [Reserved words](#).

```
% & && ( ) * + - . / ; < <= <>
= > >= ? ADD ALL ALTER
AND AS ASC BETWEEN BIGINT BOOLEAN BY
CASE CAST COLUMN COMMENT CREATE DESC DISTINCT
DISTRIBUTE DOUBLE DROP ELSE FALSE FROM FULL
GROUP IF IN INSERT INTO IS JOIN
LEFT LIFECYCLE LIKE LIMIT MAPJOIN NOT NULL
ON OR ORDER OUTER OVERWRITE PARTITION RENAME
REPLACE RIGHT RLIKE SELECT SORT STRING TABLE
THEN TOUCH TRUE UNION VIEW WHEN WHERE
```

6.1.3. Partitioned table

This topic describes the advantages and disadvantages of MaxCompute partitioned tables.

You can specify partition key columns of a partitioned table as filter conditions in WHERE clauses of SELECT statements. This improves SQL query performance and reduces costs. However, some SQL statements for partition operations are less efficient. For example, if more than 10,000 partitions exist in an output table for a job, a large number of parallel operations are performed to write data to partitions. As a result, the SQL statements may fail.

If you use a partitioned table, you must properly evaluate the number and levels of partitions based on the data source to ensure query performance and stability. MaxCompute allows a maximum of six levels of partitions in a partitioned table.

For some statements, the syntax differs between partitioned and non-partitioned tables. For more information, see [DDL statements](#) and [DML statements](#).

For more information about the statements that are used to create tables, see [Create a table](#).

6.1.4. Type conversion

6.1.4.1. Explicit conversions of data types

An explicit conversion uses CAST to convert a value type into another one. This topic describes explicit conversions.

For more information about CAST, see [CAST](#).

The following table lists the explicit conversions supported by MaxCompute SQL.

Explicit conversions

From/To	BIGINT	DOUBLE	STRING	DATETIME	BOOLEAN	DECIMAL
BIGINT	N/A	Y	Y	N	N	Y
DOUBLE	Y	N/A	Y	N	N	Y
STRING	Y	Y	N/A	Y	N	Y
DATETIME	N	N	Y	N/A	N	N
BOOLEAN	N	N	N	N	N/A	N
DECIMAL	Y	Y	Y	N	N	N/A

Y indicates that the conversion is supported. N indicates that the conversion is not supported. N/A indicates that the conversion is not required.

Note

- If the DOUBLE type is converted into the BIGINT type, the fractional part is truncated. Example: `cast(1.6 as bigint) = 1`.
- If the STRING type that meets the format of the DOUBLE type is converted into the BIGINT type, the STRING type is first converted into the DOUBLE type and then to the BIGINT type. Therefore, the fractional part is truncated. Example: `cast("1.6" as bigint) = 1`.
- If the STRING type that meets the format of the BIGINT type is converted into the DOUBLE type, one digit is retained after the decimal point. Example: `cast("1" as double) = 1.0`.
- To convert a constant string into the DECIMAL type, enclose the constant string within a pair of double quotation marks ("). If the value is not enclosed in quotation marks, it is considered a DOUBLE-type value. Example: `cast("1.234567890123456789" as decimal)`.
- An unsupported explicit conversion causes an exception.
- If a conversion fails, the system returns an error and exits.
- The conversion of the DATETIME type uses the default format `yyyy-mm-dd hh:mi:ss:ff3`. For more information, see [Conversion between string and datetime types](#).
- Some data types cannot be explicitly converted, but can be converted by using SQL built-in functions. For example, you can use the TO_CHAR function to convert the BOOLEAN type into the STRING type. For more information, see [TO_CHAR](#). You can also use the TO_DATE function to convert the STRING type into the DATETIME type.
- If the values of the DECIMAL type are not within the value range, the cast string to decimal operation may return an error, such as most significant bit overflow or least significant bit overflow truncation.

6.1.4.2. Implicit type conversion and its scope

An implicit type conversion allows MaxCompute to automatically convert data types based on the context and predefined rules. This topic describes the rules of implicit conversions.

The following table lists the rules of implicit conversions supported by MaxCompute.

Implicit conversion 1

From/To	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	FLOAT
BOOLEAN	T	F	F	F	F	F
TINYINT	F	T	T	T	T	T
SMALLINT	F	F	T	T	T	T
INT	F	F	F	T	T	T
BIGINT	F	F	F	F	T	T
FLOAT	F	F	F	F	F	T
DOUBLE	F	F	F	F	F	F
DECIMAL	F	F	F	F	F	F
STRING	F	F	F	F	F	F
VARCHAR	F	F	F	F	F	F
TIMESTAMP	F	F	F	F	F	F
BINARY	F	F	F	F	F	F

Implicit conversion 2

From/To	DOUBLE	DECIMAL	STRING	VARCHAR	TIMESTAMP	BINARY
BOOLEAN	F	F	F	F	F	F
TINYINT	T	T	T	T	F	F
SMALLINT	T	T	T	T	F	F
INT	T	T	T	T	F	F
BIGINT	T	T	T	T	F	F
FLOAT	T	T	T	T	F	F
DOUBLE	T	T	T	T	F	F
DECIMAL	F	T	T	T	F	F
STRING	T	T	T	T	F	F
VARCHAR	T	T	T	T	F	F

From/To	DOUBLE	DECIMAL	STRING	VARCHAR	TIMESTAMP	BINARY
TIMESTAMP	F	F	T	T	T	F
BINARY	F	F	F	F	F	T

T indicates that the conversion is supported. F indicates that the conversion is not supported.

 **Note**

- If an unsupported implicit conversion is performed, an error is returned.
- If a conversion fails, the system returns an error and exits.
- MaxCompute automatically performs implicit conversions on data types based on the context. If the types do not match, you can use the CAST function to explicitly convert data types.
- The rules of implicit conversions apply to specific scopes. In specific scenarios, only some rules take effect.

Implicit conversions with relational operators

Relational operators include =, <>, <, ≤, >, ≥, IS NULL, IS NOT NULL, LIKE, RLIKE, and IN. The rules for implicit conversions that use LIKE, RLIKE, and IN are different from those that use other relational operators. The rules described in this section do not apply to the three operators. The following table lists the rules of implicit conversions when relational operations are used for data of different types.

Implicit conversions with relational operators

From/To	BIGINT	DOUBLE	STRING	DATETIME	BOOLEAN	DECIMAL
BIGINT	N/A	DOUBLE	DOUBLE	N	N	DECIMAL
DOUBLE	DOUBLE	N/A	DOUBLE	N	N	DECIMAL
STRING	DOUBLE	DOUBLE	N/A	DATETIME	N	DECIMAL
DATETIME	N	N	DATETIME	N/A	N	N
BOOLEAN	N	N	N	N	N/A	N
DECIMAL	DECIMAL	DECIMAL	DECIMAL	N	N	N/A

 **Note**

- If an implicit conversion is not supported between two values that you want to compare, the relational operation cannot be performed and an error is returned.
- For more information about relational operators, see [Relational operators](#).

Implicit conversion with special relational operators

Special relational operators include LIKE, RLIKE, and IN.

Syntax of LIKE and RLIKE:

```
source like pattern;
source rlike pattern;
```

Take note of the following points for the two relational operators in implicit conversions:

- The source and pattern parameters of LIKE and RLIKE must be of the STRING type.
- Other types can neither be involved in the operation nor be implicitly converted into the STRING type.
- If the value of source or pattern is NULL, NULL is returned.

Syntax of IN:

```
key in (value1, value2,...);
```

Rules of implicit conversions for IN:

- The data types of the values on the right of the IN keyword must be consistent.
- MaxCompute automatically compares the values in the column specified by key and values on the right of the IN keyword. If these values are of the BIGINT, DOUBLE, and STRING types, the values are converted into the DOUBLE type for comparison. If these values are of the DATETIME and STRING types, the values are converted into the DATETIME type for comparison. Conversions between other types are not allowed.

Note The memory used by the compiler increases with the number of parameters used in the IN operation. If 5,000 parameters are used in an IN operation, the GCC compiler consumes 17 GB of memory for compilation. We recommend that you limit the number of parameters to 1,024. In this case, the compiler consumes a maximum of 1 GB of memory and requires 39 seconds for compilation.

Implicit conversions with arithmetic operators

Arithmetic operators include addition (+), subtraction (-), multiplication (×), division (/), and percent (%). This section describes the rules for implicit conversions that use these operators.

- Only the STRING, BIGINT, DECIMAL, and DOUBLE types can be used in arithmetic operations.
- Before an arithmetic operation, STRING data is implicitly converted into DOUBLE data.
- If an arithmetic operation involves data of the BIGINT and DOUBLE types, BIGINT data is implicitly converted into DOUBLE data.
- The DATETIME and BOOLEAN types cannot be used in arithmetic operations.

Note For more information about arithmetic operators, see [Arithmetic operators](#).

Implicit conversions with logical operators

Logical operators include AND, OR, and NOT. This section describes the rules for implicit conversions that use these operators.

- Only the BOOLEAN type can be used in logical operations.
- The other types are not supported by logical operations or implicit conversions.

Note For more information about logical operators, see [Logical operators](#).

6.1.4.3. SQL built-in functions

MaxCompute SQL provides a variety of built-in functions. These functions can be used to calculate one or more columns of a row and provide data of any type.

The following rules apply to implicit conversions:

- If you call a function and the data type of an input parameter is different from that defined in the function, the data type of the input parameter is converted into the function-defined data type.
- The parameters of each built-in function in MaxCompute SQL have different requirements for implicit

conversions. For more information, see [Built-in functions](#).

6.1.4.4. CASE WHEN

This topic describes the implicit conversion rules for CASE WHEN.

Implicit conversion rules for CASE WHEN:

- If return values are of the BIGINT and DOUBLE types, all the values are converted into the DOUBLE type.
- If return values include those of the STRING type, all the values are converted into the STRING type. If a conversion, such as conversion from BOOLEAN into STRING, fails, an error is returned.
- Conversions between other types are not allowed.

6.1.4.5. Conversions between the STRING and DATETIME types

MaxCompute supports conversions between the STRING and DATETIME types.

The format used in conversions is yyyy-mm-dd hh:mi:ss.ff3.

Valid values of time units

Time unit	String (not case-sensitive)	Valid value
Year	yyyy	0001 to 9999
Month	mm	01 to 12
Day	dd	01 to 31
Hour	hh	00 to 23
Minute	mi	00 to 59
Second	ss	00 to 59
Millisecond	ff3	00 to 999

Note

- If the first digit of a valid value for each time unit is 0, 0 cannot be omitted. For example, 2017-1-9 12:12:12 is an invalid DATETIME format and it cannot be converted from the STRING type into the DATETIME type. It must be written as 2017-01-09 12:12:12.
- Only the STRING type that meets the preceding format requirements can be converted into the DATETIME type. For example, `cast("2017-12-31 02:34:34" as datetime)` converts 2017-12-31 02:34:34 of the STRING type into the DATETIME type. Similarly, if the DATETIME type is converted into the STRING type, the default conversion format is yyyy-mm-dd hh:mi:ss. Conversions that use the following or similar statements may fail, and errors are returned.

```
cast("2017/12/31 02/34/34" as datetime)
cast("20171231023434" as datetime)
cast("2017-12-31 2:34:34" as datetime)
```

MaxCompute provides the TO_DATE function to convert the STRING type that does not meet the DATETIME format requirements into the DATETIME type. For more information, see [TO_DATE](#).

6.2. Operators

6.2.1. Relational operators

This topic describes relational operators in MaxCompute SQL operators.

Relational operators

Operator	Description
A=B	If A or B is NULL, NULL is returned. If A is equal to B, TRUE is returned. Otherwise, FALSE is returned.
A<>B	If A or B is NULL, NULL is returned. If A is unequal to B, TRUE is returned. Otherwise, FALSE is returned.
A<B	If A or B is NULL, NULL is returned. If A is less than B, TRUE is returned. Otherwise, FALSE is returned.
A<=B	If A or B is NULL, NULL is returned. If A is less than or equal to B, TRUE is returned. Otherwise, FALSE is returned.
A>B	If A or B is NULL, NULL is returned. If A is greater than B, TRUE is returned. Otherwise, FALSE is returned.
A>=B	If A or B is NULL, NULL is returned. If A is greater than or equal to B, TRUE is returned. Otherwise, FALSE is returned.
A IS NULL	If A is NULL, TRUE is returned. Otherwise, FALSE is returned.
A IS NOT NULL	If A is not NULL, TRUE is returned. Otherwise, FALSE is returned.
A LIKE B	<p>If A or B is NULL, NULL is returned. A is a string and B is the pattern that you want to match. If A matches B, TRUE is returned. Otherwise, FALSE is returned. The percent sign (%) is a wildcard character that matches an arbitrary number of characters. The underscore (_) is a wildcard character that matches a single character. To use these two characters as regular characters, use backslashes (\) to escape them: \% and _.</p> <pre>'aaa'like 'a' = TRUE 'aaa'like'a%' = TRUE 'aaa'like'aab' = FALSE 'a%b'like'a\b' = TRUE 'axb'like'a\b' = FALSE</pre>
A RLIKE B	If A or B is NULL, NULL is returned. A is a string and B is a regular expression consisting of string constants. If A matches B, TRUE is returned. Otherwise, FALSE is returned. If B is an empty string, the system returns an error and exits.
A IN B	B is a set. If A is NULL, NULL is returned. If A is in B, TRUE is returned. Otherwise, FALSE is returned. If B contains only one element NULL, namely, A IN (NULL), NULL is returned. If B contains NULL, the type of NULL is considered the same as the other elements in B. B must be a constant and have at least one element. All elements must be of the same type.

The values of the DOUBLE type in MaxCompute are different in precision. We recommend that you do not use the equal sign (=) for comparison between two values of the DOUBLE type. You can deduct a value of the DOUBLE type from the other value of the DOUBLE type and use the absolute value of the difference to determine whether the two values are equal. If the absolute value is negligible, the two values of the DOUBLE type are considered equal. Example:

```
abs(0.9999999999 - 1.0000000000) < 0.0000000001
-- 0.9999999999 and 1.0000000000 have 10 decimal digits, whereas 0.0000000001 has 9 decimal digits.
-- 0.9999999999 is considered equal to 1.0000000000.
```

 **Note**

- ABS is a built-in function provided by MaxCompute to take the absolute value of its input. For more information, see [ABS](#).
- In most cases, a value of the DOUBLE type in MaxCompute can retain 16 valid digits.

6.2.2. Arithmetic operators

This topic describes arithmetic operators in MaxCompute SQL operators.

Arithmetic operators

Operator	Description
A + B	If A or B is NULL, NULL is returned. Otherwise, the result of A + B is returned.
A - B	If A or B is NULL, NULL is returned. Otherwise, the result of A - B is returned.
A * B	If A or B is NULL, NULL is returned. Otherwise, the result of A × B is returned.
A / B	If A or B is NULL, NULL is returned. Otherwise, the result of A/B is returned. If both A and B are of the BIGINT type, a value of the DOUBLE type is returned.
A % B	If A or B is NULL, NULL is returned. Otherwise, the result of A % B is returned.
+A	A is returned.
-A	If A is NULL, NULL is returned. Otherwise, -A is returned.

 **Note**

- You can use only the values of the STRING, BIGINT, DOUBLE, or DECIMAL type to perform arithmetic operations. You cannot use the values of the DATETIME or BOOLEAN type to perform arithmetic operations.
- Values of the STRING type are implicitly converted into those of the DOUBLE type before arithmetic operations.
- If you use the values of the BIGINT and DOUBLE types to perform arithmetic operations, the value of the BIGINT type is implicitly converted into that of the DOUBLE type before arithmetic operations. A value of the DOUBLE type is returned.
- If both A and B are of the BIGINT type, a value of the DOUBLE type is returned after you perform the A/B operation. For other arithmetic operations, a value of the BIGINT type is returned.

6.2.3. Bitwise operators

This topic describes bitwise operators in MaxCompute SQL operators.

Bitwise operators

Operator	Description
A & B	The bitwise AND result of A and B is returned. For example, the result of 1&2 is 0, and the result of 1&3 is 1. The bitwise AND result of NULL and a value is NULL. Both A and B must be of the BIGINT type.
A B	The bitwise OR result of A and B is returned. For example, the result of 1 2 is 3, and the result of 1 3 is 3. The bitwise OR result of NULL and a value is NULL. Both A and B must be of the BIGINT type.

 **Notice** Bitwise operators do not support implicit type conversions. You can use the values only of the BIGINT type in bitwise operations.

6.2.4. Logical operators

This topic describes logical operators in MaxCompute SQL.

Logical operators

Operator	Description
A and B	TRUE and TRUE = TRUE
	TRUE and FALSE = FALSE
	FALSE and TRUE = FALSE
	FALSE and NULL = FALSE
	FALSE and FALSE = FALSE
	NULL and FALSE = FALSE
	TRUE and NULL = NULL
	NULL and TRUE = NULL
	NULL and NULL = NULL
A or B	TRUE or TRUE = TRUE
	TRUE or FALSE = TRUE
	FALSE or TRUE = TRUE
	FALSE or NULL = NULL
	NULL or FALSE = NULL
	TRUE or NULL = TRUE
	NULL or TRUE = TRUE
	NULL or NULL = NULL
NOT A	If A is NULL, NULL is returned.
	If A is TRUE, FALSE is returned.

Operator	Description
	If A is FALSE, TRUE is returned.

 **Note** Logical operators do not support implicit type conversions. You can use only the values of the BOOLEAN type in logical operations.

6.3. LOAD

This topic describes how to use LOAD statements to import data from external storage, such as Object Storage Service (OSS), to a table or a table partition in MaxCompute.

Usage notes

MaxCompute must be authorized to access data in external storage. The methods used to authorize MaxCompute to access external data by using LOAD statements are the same as those used to authorize MaxCompute external tables. You can use one of the following methods for authorization:

- Specify the AccessKey ID and AccessKey secret in the directory in which data is stored. In this example, a directory on OSS is used.

```
'oss://<yourAccessKeyId>:<yourAccessKeySecret>@oss-cn-hangzhou-zmf.aliyuncs.com/my_bucket_id/my_location/'
```

 **Notice** If you use this method, the AccessKey ID and AccessKey secret are displayed in plaintext. This may pose security risks. We recommend that you do not use this method.

- Use Security Token Service (STS).

Use an extractor or a storage handler to import data

Syntax:

```
LOAD OVERWRITE|INTO TABLE table_name [PARTITION(partcol1=val1, partcol2=val2 ...)]
FROM LOCATION external_location
[STORED BY StorageHandler]
[WITH SERDEPROPERTIES (Options)];
```

Parameters:

- **table_name**: the name of the destination table into which you want to insert data. You must create a destination table before you insert data into it. The schema of the destination table must be the same as the format of the external data.
- **LOAD INTO**: inserts data into a table or partition.
- **LOAD OVERWRITE**: clears the existing data from a table or partition and inserts data into the table or partition.
- **SORTED BY**: specifies the name of the storage handler. You can use this clause in the same way as you use it for MaxCompute external tables.
- **LOCATION**: specifies the OSS directory in which the data files you want to read are saved. The system reads all files in the directory by default.
- **WITH SERDEPROPERTIES**: specifies the properties of the MaxCompute external table. The properties supported by WITH SERDEPROPERTIES are the same as those supported by the MaxCompute external table.

Assume that the vehicle.csv file contains the following data:

```

1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N

```

Examples:

```

oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/data_location/
-- Save the vehicle.csv file to a folder in an OSS bucket. Obtain the structure of the directory where
the MaxCompute external table is saved.
CREATE TABLE ambulance_data_csv_load (
vehicleId INT,
recordId INT,
patientId INT,
calls INT,
locationLatitude DOUBLE,
locationLongitude DOUBLE,
recordTime STRING,
direction STRING );
-- Create a destination table named ambulance_data_csv_load.
LOAD OVERWRITE TABLE ambulance_data_csv_load
FROM
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/bucket/data_location/'
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole', -- The Alibaba Cloud Resource
Name (ARN) of the AliyunODPSDefaultRole role.
'odps.text.option.delimiter'=', '
);
-- Import the CSV file from OSS to the destination table.

```

Import data stored in an open source format

Syntax:

```

LOAD OVERWRITE|INTO TABLE table_name [PARTITION(partcol1=val1, partcol2=val2 ...)]
FROM LOCATION external_location
[ROW FORMAT SERDE '<serde class>'
 [WITH SERDEPROPERTIES ('odps.properties.rolearn'='${roleran}' [, 'name2'='value2',...])]
]
STORED AS <file format>;

```

Parameters:

- **table_name**: the name of the destination table into which you want to insert data. You must create a destination table before you insert data into it. The schema of the destination table must be the same as the format of the external data.
- **LOAD INTO**: inserts data into a table or partition.
- **LOAD OVERWRITE**: clears the existing data from a table or partition and inserts data into the table or partition.
- **STORED AS**: specifies the format of the imported data file. Valid values include ORC, PARQUET, RCFILE,

SEQUENCEFILE, and TEXTFILE. You can use this clause in the same way as you use it for MaxCompute external tables.

- **ROW FORMAT SERDE:** This clause is optional. You can use this clause in the same way as you use it for MaxCompute external tables. Mappings between file formats and SerDe classes:
 - SEQUENCEFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
 - TEXTFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
 - RCFILE: org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe
 - ORC: org.apache.hadoop.hive.ql.io.orc.OrcSerde
 - ORCFILE: org.apache.hadoop.hive.ql.io.orc.OrcSerde
 - PARQUET: org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
 - AVRO: org.apache.hadoop.hive.serde2.avro.AvroSerDe
- **SERDEPROPERTIES:** If you use STS to grant MaxCompute the required permissions, you must use this parameter to specify the value of the odps.properties.rolearn property. This property indicates the ARN of the AliyunODPSDefaultRole role. If the owner of MaxCompute and that of OSS use the same account, one-click authorization is allowed.

Examples:

```
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/data_location/ds=20200910/'
-- Assume that the OSS directory where the MaxCompute external table is saved uses the preceding structure. The MaxCompute external table in the directory does not include partition key columns. The information of partition key columns is included in the directory information.
CREATE TABLE ambulance_data_csv_load_pt (
  vehicleId STRING,
  recordId STRING,
  patientId STRING,
  calls STRING,
  locationLatitude STRING,
  locationLongitude STRING,
  recordTime STRING,
  direction STRING)
PARTITIONED BY (ds STRING);
-- Create a destination table named ambulance_data_csv_load_pt.
LOAD OVERWRITE TABLE ambulance_data_csv_load_pt PARTITION (ds='20200910')
FROM
LOCATION 'oss://<yourAccessKeyId>:<yourAccessKeySecret>@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/data_location/'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
STORED AS TEXTFILE;
-- Import data stored in open source formats from OSS to the destination table.
```

 **Note** If you download data to a partitioned table, the schema of the external table in the OSS directory does not include partition key columns. In the preceding example, the external table in the directory contains only the business fields. The table does not include partition key columns.

6.4. DDL statements

6.4.1. Table operations

6.4.1.1. Create a table

This topic describes how to use a DDL statement to create a table.

Syntax:

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [DEFAULT value] [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY | RANGE CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [,
col_name [ASC | DESC] ...))] INTO number_of_buckets BUCKETS] -- Specify the shuffle and sort attrib
utes when you create a clustered table.
[STORED BY StorageHandler] -- Used only for external tables.
[WITH SERDEPROPERTIES (Options)] -- Used only for external tables.
[LOCATION OSSLocation] -- Used only for external tables.
[STORED AS AliOrc]-- Specify the storage format of the table. You can specify AliORC only for newly
created internal tables.
[TBLPROPERTIES("compressionstrategy"="normal/high/extreme",-- Specify the compression policy for dat
a storage.
"transactional"="true")] -- Specify a transactional table. You can then execute UPDAT
E or DELETE statements on the table.
[LIFECYCLE days];
CREATE TABLE [IF NOT EXISTS] table_name
[AS select_statement | LIKE existing_table_name];
```

Parameters:

- **table_name** and **col_name**: the table name and column name. The names are not case-sensitive. The names cannot contain special characters. A table or column name can contain only letters, digits, and underscores (_) and can be up to 128 bytes in length. We recommend that you start the table or column name with a letter. A table can contain a maximum of 1,200 columns.
- **data_type**: the type of data. Valid values: BIGINT, DOUBLE, BOOLEAN, DATETIME, DECIMAL, STRING, ARRAY<T>, and MAP<T1,T2>.

 **Note** If you want to use new data types, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, and BINARY, you must enable these data types.

- To enable new data types at the session level, add `set odps.sql.type.system.odps2=true;` before the SQL statement. Then, commit them for execution.
- To enable new data types at the project level, run the following command as the project owner:

```
setproject odps.sql.type.system.odps2=true;
```

- **DEFAULT value**: the default value of the column. If you do not specify the value of the column in an INSERT operation, the default value is used for the column.
- **COMMENT table_comment**: the comment of the table. The comment must be a valid string that can be up to 1,024 bytes in length.
- **PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)**: the partition field of the table. The following data types are supported: TINYINT, SMALLINT, INT, BIGINT, VARCHAR, and STRING.

A partition value cannot contain double-byte characters. It must start with a letter, followed by letters, digits, or special characters. The value can be up to 128 bytes in length. The value can contain the following special characters: spaces, colons (:), underscores (_), dollar signs (\$), number signs (#), periods (.), exclamation points (!), and at signs (@). Other characters, such as \t, \n, and /, are considered undefined characters. After you use partition fields to partition a table, a full table scan is not triggered when you add partitions, update partition data, or read partition data. This makes data processing more efficient.

 **Note** A table can contain a maximum of 60,000 partitions at six or lower levels.

- **CLUSTERED BY | RANGE CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC | DESC] ...]) INTO number_of_buckets BUCKETS:** specifies the shuffle and sort attributes when you create a clustered table. Clustered tables are classified into hash-clustered tables and range-clustered tables.
 - Hash-clustered tables
 - **CLUSTERED BY:** the hash key. MaxCompute performs hash operations on the specified columns and distributes data to each bucket based on the hash values. We recommend that you specify a column that has large value ranges and a small number of duplicate key-value pairs in **CLUSTERED BY**. This prevents data skew and hot spots and makes concurrent execution more efficient. To optimize JOIN operations, we recommend that you select commonly used join or aggregation keys. The join and aggregation keys are similar to the primary keys in conventional databases.
 - **SORTED BY:** specifies how the data in a bucket is sorted. To achieve better performance, we recommend that you set **SORTED BY** and **CLUSTERED BY** to the same value. If you specify the **SORTED BY** clause, MaxCompute automatically generates indexes and uses these indexes to speed up your queries.
 - **INTO number_of_buckets BUCKETS:** the number of hash buckets. This parameter is required and varies based on your data volume. By default, MaxCompute supports a maximum of 1,111 reducers. Therefore, MaxCompute supports a maximum of 1,111 hash buckets. You can run the `set odps.sql.reducer.instances=xxx;` command to increase the number of hash buckets. However, the maximum number of hash buckets is 4,000. If the value exceeds 4000, computing performance deteriorates.

 **Note** We recommend that you comply with the following rules when you specify the number of hash buckets:

- Keep the size of each bucket around 500 MB. For example, if you want to add 1,000 buckets to a partition whose size is 500 GB, the size of each bucket is 500 MB on average. If a table contains large amounts of data, you can increase the size of each bucket from 500 MB to a size in the range of 2 GB to 3 GB. You can also run the `set odps.sql.reducer.instances=xxx;` command to set the number of hash buckets to a value greater than 1111.
- To optimize JOIN operations, we recommend that you do not specify the shuffle and sort attributes. The number of hash buckets in one table must be the multiple of that in the other table. For example, one table has 256 hash buckets and the other table has 512 hash buckets. We recommend that you set the number of hash buckets to 2^n , such as 512, 1,024, 2,048, or 4,096. This way, MaxCompute can automatically split and merge hash buckets. To make the execution more efficient, we recommend that you do not specify the shuffle and sort attributes.

- Range-clustered tables
 - RANGE CLUSTERED BY: the column for range clustering. MaxCompute performs bucket operations on the specified columns and distributes data to each bucket based on the bucket numbers.
 - SORTED BY: specifies how the data in a bucket is sorted. You can use this clause in the same way you use it for a hash-clustered table.
 - INTO number_of_buckets BUCKETS: the number of hash buckets. Compared with hash-clustered tables, range-clustered tables have no limits on the number of buckets if the data is reasonably distributed. If you do not specify the number of buckets for a range-clustered table, MaxCompute automatically determines the optimal number of buckets based on your data volume.

 **Note** If join and aggregate operations are performed on range-clustered tables and the join or group key is the range clustering key or the prefix of the range clustering key, you can control flags to remove data shuffling. This makes the execution more efficient. You can use the `set odps.optimizer.enable.range.partial.repartitioning` flag to control data shuffling. Valid values are true and false. The default value is false, which indicates that data shuffling is disabled.

- STORED AS: the storage format of the table. The default value is CFile2. AliORC in C++ is now available. It is developed by the MaxCompute storage team. AliORC is fully compatible with the open source Optimized Row Columnar (ORC). Compared with CFile2, AliORC frees up more than 10% of storage and improves read performance by more than 20%.
- LIFECYCLE: the lifecycle of the table, in days. If you execute the CREATE TABLE LIKE statement to create a table based on the schema of the source table, the lifecycle settings of the source table are not replicated.
- CREATE TABLE [IF NOT EXISTS] table_name [AS select_statement | LIKE existing_table_name]:
 - If you execute the `CREATE TABLE...AS select_statement...` statement to create a table, the data is replicated from the source table. However, the `CREATE TABLE...AS select_statement...` statement does not replicate the partition attributes of the source table. Partition key columns in the source table become standard columns in the destination table.
 - If you execute the `CREATE TABLE...LIKE existing_table_name` statement to create a table, the destination table has the same schema as the source table. However, the statement does not replicate data or lifecycle settings from the source table to the destination table.
- Clauses related to external tables:
 - STORED BY StorageHandler: the storage handler that is specified based on the format of data in the external table.
 - WITH SERDEPROPERTIES (Options): the parameters for authorization, compression, and character parsing in the external table.
 - LOCATION OSSLocation: the Object Storage Service (OSS) directory where the external table is saved.
- TBLPROPERTY("compressionstrategy"="xxx"): the attribute of the transactional table. You can specify the lifecycle and comment attributes in this clause. compressionstrategy is the compression policy for data storage. The valid values are normal, high, and extreme. The compressionstrategy parameter must be in lowercase. However, the value of this parameter is not case-sensitive. This attribute applies only to the internal tables of MaxCompute.

The default compression policy is normal for non-archived data and high for archived data.

- Definitions of the valid values:
 - normal: indicates zstd level 1.
 - high: indicates zstd level 4.
 - extreme: indicates zstd level 9.

- You can also specify the compression policy for a project. The project owner must use `odps.storage.compression.strategy` to specify the compression policy in the common attributes for project management. After this attribute is specified, the configuration applies to all generated non-archived data and does not affect the archive policy.
- `TBLPROPERTIES("transactional"="true")`: indicates that the table to create is a transactional table. You can execute the `UPDATE` or `DELETE` statement to update or delete data by row.

Note If you do not specify `IF NOT EXISTS` in a table creation statement and another table with the same name exists, an error is returned. If you specify `IF NOT EXISTS` in a table creation statement, a success message is returned. The message is returned regardless of whether a table with the same name exists. The message is returned even if the schema of the existing table is different from that of the table to create. In addition, the metadata of the existing table remains unchanged.

Examples for creating a partitioned table

The following example shows how to create a table named `sale_detail` to store sales records. The table has two partition key columns: `sale_date` and `region`. Syntax:

```
CREATE TABLE IF NOT EXISTS sale_detail
(
  shop_name      STRING,
  customer_id    STRING,
  total_price    DOUBLE
)
PARTITIONED BY (sale_date STRING, region STRING);
-- Create a partitioned table named sale_detail.
```

CREATE TABLE...AS... example

You can execute the `CREATE TABLE...AS select_statement...` statement to create a table for which data is replicated from the source table. Example:

```
CREATE TABLE sale_detail_ctas1 AS
SELECT * FROM sale_detail;
```

If the `sale_detail` table contains data, all data in the `sale_detail` table is replicated to the `sale_detail_ctas1` table after you execute the preceding statement.

Note The `sale_detail` table is a partitioned table. When you execute the `CREATE TABLE...AS select_statement...` statement to create `sale_detail_ctas1`, partition attributes are not replicated and partition key columns in the `sale_detail` table are considered standard columns in the `sale_detail_ctas1` table. `sale_detail_ctas1` is a non-partitioned table that has five columns.

If the `SELECT` clause in the `CREATE TABLE...AS select_statement...` statement uses constants as column values, we recommend that you specify the names of columns. Example:

```
CREATE TABLE sale_detail_ctas2
AS
SELECT shop_name, customer_id, total_price, '2013' AS sale_date, 'China' AS region
FROM sale_detail;
```

Example without column aliases specified:

```
CREATE TABLE sale_detail_ctas3
AS
SELECT shop_name, customer_id, total_price, '2013', 'China'
FROM sale_detail;
```

CREATE TABLE...LIKE... example

If you want the source and destination tables to have the same schema, you can execute the `CREATE TABLE...LIKE...` statement. Example:

```
CREATE TABLE sale_detail_like LIKE sale_detail;
```

The schema of `sale_detail_like` is exactly the same as that of `sale_detail`. The tables have the same attributes, such as column names, column comments, and table comments, except for the lifecycle. However, data in `sale_detail` is not replicated to `sale_detail_like`.

Create clustered tables

Create hash-clustered tables:

```
CREATE TABLE T1 (a STRING, b STRING, c BIGINT) CLUSTERED BY (c) SORTED BY (c) INTO 1024 BUCKETS; --
Create a hash-clustered non-partitioned table.
CREATE TABLE T1 (a STRING, b STRING, c BIGINT) PARTITIONED BY (dt STRING) CLUSTERED BY (c) SORTED BY
(c) INTO 1024 BUCKETS; -- Create a hash-clustered partitioned table.
```

Create range-clustered tables:

```
CREATE TABLE T2 (a STRING, b STRING, c BIGINT) RANGE CLUSTERED BY (c) SORTED BY (c) INTO 1024 BUCKET
S; -- Create a range-clustered non-partitioned table.
CREATE TABLE T2 (a STRING, b STRING, c BIGINT) PARTITIONED BY (dt STRING) CLUSTERED BY (c) SORTED BY
(c) ; -- Create a range-clustered partitioned table. The number of buckets can be empty.
```

Create transactional tables

```
-- Create a non-partitioned transactional table.
create table txn_table(id bigint) tblproperties("transactional"="true");
-- Create a partitioned transactional table.
create table if not exists txn_table_pt(id bigint) partitioned by(ds string) tblproperties ("transac
tional"="true");
```

Query table information

MaxCompute allows you to execute the `DESC` statement to query table information.

```
DESC <table_name>;
-- Query table information.
DESC EXTENDED <table_name>;
-- Query information about an external table.
-- You can also check whether the table is a transactional table. If you want to check whether the t
able is a transactional table, you must upgrade the MaxCompute client to 0.35.0.
```

Query the statements that are used to create tables

MaxCompute allows you to execute the `SHOW CREATE TABLE` statement to query the statements that are used to create tables.

```
SHOW CREATE TABLE <table_name>;
```

Note You can use this statement to generate SQL DDL statements that are used to create tables. These DDL statements can be used to rebuild the table schema.

6.4.1.2. Delete a table

This topic describes how to use a DDL statement to delete a table.

Syntax:

```
DROP TABLE [IF EXISTS] table_name;
```

Note If you do not specify IF EXISTS and the table does not exist, an error is returned. If this option is specified, a success message is returned regardless of whether the table exists.

Examples:

```
CREATE TABLE sale_detail_drop LIKE sale_detail;
DROP TABLE sale_detail_drop;
-- If the table exists, a success message is returned. If the table does not exist, an error is returned.
DROP TABLE IF EXISTS sale_detail_drop2;
-- A success message is returned regardless of whether the sale_detail_drop2 table exists.
```

6.4.1.3. Rename a table

This topic describes how to use a DDL statement to rename a table.

Syntax:

```
ALTER TABLE table_name RENAME TO new_table_name;
```

- Note**
- The RENAME operation changes only the name of a table. Data in the table remains unchanged.
 - If a table that has the same name as the table specified by new_table_name exists, an error is returned.
 - If the table specified by table_name does not exist, an error is returned.

Examples:

```
CREATE TABLE sale_detail_rename1 LIKE sale_detail;
ALTER TABLE sale_detail_rename1 RENAME TO sale_detail_rename2;
```

6.4.1.4. Change the owner of a table

This topic describes how to use a DDL statement to change the owner of a table.

MaxCompute SQL allows you to execute the `CHANGEOWNER` statement to change the owner of a table.

Syntax:

```
ALTER TABLE table_name CHANGEOWNER TO 'ALIYUN$xxx@aliyun.com';
```

6.4.1.5. Modify the comment of a table

This topic describes how to use a DDL statement to modify the comment of a table.

Syntax:

```
ALTER TABLE table_name SET COMMENT 'tbl comment';
```

Note

- The table specified by table_name must exist.
- The value of comment can be up to 1,024 bytes.

Examples:

```
ALTER TABLE sale_detail SET COMMENT 'new coments for table sale_detail';
```

You can execute the DESC statement in MaxCompute to view the comment of the table after modification.

6.4.1.6. Modify the lifecycle settings of a table

MaxCompute provides the lifecycle management feature to facilitate storage release and simplify data reclamation. This topic describes how to use a DDL statement to modify the lifecycle settings of a table.

Syntax:

```
ALTER TABLE table_name SET lifecycle days;
```

Note

- days: the lifecycle duration, which must be a positive integer in days.
- table_name: the name of the table whose lifecycle settings you want to modify.
- You can specify the lifecycle when you create a table. The lifecycle can be specified only for a table rather than a partition. After a lifecycle is specified for a partitioned table, the lifecycle applies to the partitions of the table.
- If data in a table is modified, the value of LastDataModifiedTime is updated. Therefore, MaxCompute determines whether to reclaim this table based on the value of LastDataModifiedTime and the lifecycle settings.
- For a non-partitioned table, if the data remains unchanged for the duration specified by days after data is last modified, MaxCompute automatically executes a statement such as DROP TABLE to reclaim the table.
- For a partitioned table, MaxCompute determines whether to reclaim a partition based on the value of LastDataModifiedTime for the partition. Unlike non-partitioned tables, a partitioned table is not deleted even if all of the partitions are reclaimed.
- You can only modify the lifecycle settings instead of disabling the lifecycle for non-partitioned tables. You can disable the lifecycle for a specific partition in a partitioned table.

Examples:

```
create table test_lifecycle(key string) lifecycle 100;
-- Create a table named test_lifecycle with a lifecycle of 100 days.
alter table test_lifecycle set lifecycle 50;
-- Change the lifecycle of the test_lifecycle table to 50 days.
```

6.4.1.7. Disable or enable the lifecycle feature

In some cases, if you do not want some partitions to be automatically reclaimed based on the lifecycle feature, you can disable the lifecycle feature for these partitions. This topic describes how to use a DDL statement to disable or enable the lifecycle feature.

Syntax:

```
ALTER TABLE table_name partition[partition_spec] ENABLE|DISABLE LIFECYCLE;
```

Note

- **DISABLE LIFECYCLE:** disables the lifecycle feature for a table or partition.
 - It prevents the reclamation of a table and its partitions based on the lifecycle feature. This parameter has a higher priority than partition_spec enable lifecycle. If you use table disable lifecycle, partition_spec enable lifecycle is invalid.
 - After the lifecycle feature of a table is disabled, the table lifecycle settings and the ENABLE and DISABLE tags of partitions in the table are retained.
 - After the lifecycle feature of a table is disabled, you can still modify the lifecycle settings of the table and its partitions.
- **ENABLE LIFECYCLE:** enables the lifecycle feature for a table or partition.
 - A table and its partitions can be reclaimed based on the lifecycle feature again. The lifecycle settings of the current table and its partitions are used by default.
 - Before you enable the lifecycle feature for a table, you can modify the lifecycle settings of the table and its partitions. This prevents data from being mistakenly reclaimed due to the use of previous settings.

Examples:

```
ALTER TABLE trans DISABLE LIFECYCLE;
-- Disable the lifecycle feature for the trans table.
ALTER TABLE trans PARTITION(dt='20141111') DISABLE LIFECYCLE;
-- Disable the lifecycle feature for the dt='20141111' partition in the trans table.
ALTER TABLE trans ENABLE LIFECYCLE;
-- Enable the lifecycle feature for the trans table.
ALTER TABLE trans PARTITION(dt='20141111') ENABLE LIFECYCLE;
-- Enable the lifecycle feature for the dt='20141111' partition in the trans table.
```

6.4.1.8. Modify the value of LastDataModifiedTime for a table

MaxCompute SQL allows you to perform the TOUCH operation to modify the value of LastDataModifiedTime for a table. This operation changes the value of LastDataModifiedTime for a table to the current time. This topic describes how to use a DDL statement to modify the value of LastDataModifiedTime for a table.

Syntax:

```
ALTER TABLE table_name TOUCH;
```

Note

- If the table specified by `table_name` does not exist, an error is returned.
- This operation modifies the value of `LastDataModifiedTime` for a table. In this case, MaxCompute considers a change to the table data and recalculates the lifecycle.

6.4.1.9. Modify the clustering attributes of a table

This topic describes how to use a DDL statement to modify the clustering attributes of a table.

MaxCompute allows you to add or remove the clustering attributes of a partitioned table by using the `ALTER TABLE` statement.

Note

- The `ALTER TABLE` statement can modify only the clustering attributes of a partitioned table. The clustering attributes of a non-partitioned table cannot be modified after these attributes are added. The `ALTER TABLE` statement is applicable to the tables that already exist. After new clustering attributes are added, new partitions are stored based on the modified clustering attributes.
- The `ALTER TABLE` statement affects only the new partitions of partitioned tables, including the partitions generated by using the `INSERT OVERWRITE` statement. New partitions are stored based on the new clustering attributes, while the clustering attributes and storage of existing partitions remain unchanged. After you specify the clustering attributes for a table, you can remove the clustering attributes and add clustering attributes for the table again. You can specify different clustering columns, sort columns, and numbers of buckets for new partitions.
- The `ALTER TABLE` statement takes effect only on the new partitions of a table. Therefore, you cannot specify a partition in this statement.

Add hash clustering attributes to a table

Syntax:

```
ALTER TABLE table_name
[CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC | DESC]
] ...))] INTO number_of_buckets BUCKETS]
```

Remove hash clustering attributes from a table

Syntax:

```
ALTER TABLE table_name NOT CLUSTERED;
```

Add range clustering attributes to a table

If you do not specify the number of buckets, MaxCompute automatically determines the optimal number based on the data volume.

Syntax:

```
ALTER TABLE table_name
[RANGE CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC | DESC]
] ...))] INTO number_of_buckets BUCKETS]
```

Remove range clustering attributes from a table or partition

Syntax:

```
ALTER TABLE table_name NOT CLUSTERED;  
ALTER TABLE table_name partition_spec NOT CLUSTERED;
```

6.4.1.10. Delete data from a non-partitioned table

This topic describes how to use a DDL statement to delete data from a non-partitioned table.

Syntax:

```
TRUNCATE TABLE table_name;
```

 **Note** This statement is used to delete data from a specific non-partitioned table. To delete data from a partitioned table, execute the `ALTER TABLE table_name DROP PARTITION(partition_spec)` statement.

6.4.1.11. Archive table data

This topic describes how to use a DDL statement to archive the data of a table.

If a project does not have enough space and data needs to be compressed or deleted, you can use table archiving of MaxCompute to compress data by about 50%. Table archiving uses a compression algorithm with a high compression ratio. It saves data as redundant array of independent disks (RAID) files. Data is no longer simply stored in three copies. Instead, six copies and three check blocks are maintained to increase the compression ratio from 1:3 to 1:1.5. Table archiving allows data to consume only half of the physical space.

However, this feature comes at a price. If a data block or machine is faulty, it requires a long time to restore the data block, and read performance deteriorates. Therefore, this feature is suitable for the compression of cold data. For example, you can use this feature to store a large number of logs that are less used as RAID files for a long time.

Syntax:

```
ALTER TABLE [table_name] <PARTITION(partition_name='partition_value')> ARCHIVE;
```

 **Note**

- If the table specified by table_name does not exist, an error is returned.
- If the partition specified by partition_name does not exist, an error is returned.

Examples:

```
alter table my_log partition(ds='20170101') archive;
```

Output:

```
Summary:  
table name: test0128 /pt=a instance count: 1 run time: 21  
before merge, file count: 1 file size: 456 file physical size: 1368  
after merge, file count: 1 file size: 512 file physical size: 768
```

Note

The output shows the changes in the logical size and physical size after the data is archived. In the archiving process, multiple small files are automatically merged. After the data is archived, you can execute the `DESC EXTENDED` statement to check whether the data in the partition has been archived and view the usage of physical space:

```
desc extended my_log partition(ds='20170101');
-- The following result is returned:
+-----+
PartitionSize: 512 |
+-----+
CreateTime: 2017-01-28 07:05:20 |
LastDDLTime: 2017-01-28 07:05:20 |
LastModifiedTime: 2017-01-28 07:05:21 |
+-----+
```

6.4.1.12. Forcibly delete data from a table or partition

This topic describes how to use a DDL statement to forcibly delete data from a table or partition.

If you want to forcibly and irrecoverably delete data from a table or partition to immediately release storage, you can execute a DROP statement with the PURGE option.

Syntax:

```
DROP TABLE tblname PURGE;
ALTER TABLE tblname DROP PARTITION(part_spec) PURGE;
```

Examples:

```
drop table my_log purge;
alter table my_log drop partition (ds='20170618') purge;
```

6.4.2. View-based operation

6.4.2.1. Create a view

This topic describes how to use a DDL statement to create a view.

Syntax:

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS] view_name
[(col_name [COMMENT col_comment], ...)]
[COMMENT view_comment]
[AS select_statement];
```

Note

- To create a view, you must have read permissions on the table referenced by the view.
- A view can contain only one valid SELECT statement.
- A view can reference other views but cannot reference itself. Circular references are not supported.
- You cannot write data to a view. For example, the `INSERT INTO` and `INSERT OVERWRITE` statements do not work on views.
- If the table referenced by a view changes, you may no longer be able to access the view. For example, a view becomes inaccessible after the table it references is deleted. You must maintain the mappings between referenced tables and views.
- If the `CREATE VIEW` statement is executed without the IF NOT EXISTS option and the view already exists, an exception is returned. In this case, you can execute `CREATE VIEW` or `REPLACE VIEW` to recreate the view. The permissions on the view remain unchanged after the view is recreated.

Examples:

```
create view if not exists sale_detail_view
(store_name, customer_id, price, sale_date, region)
comment 'a view for table sale_detail'
as select * from sale_detail;
-- Create a view named sale_detail_view.
```

6.4.2.2. Delete a view

This topic describes how to use a DDL statement to delete a view.

Syntax:

```
DROP VIEW [IF EXISTS] view_name;
```

- Note** If the partition does not exist and IF EXISTS is not specified, an error is returned.

Examples:

```
DROP VIEW IF EXISTS sale_detail_view;
```

6.4.2.3. Rename a view

This topic describes how to use a DDL statement to rename a view.

Syntax:

```
ALTER VIEW view_name RENAME TO new_view_name;
```

- Note** If a view with the same name already exists, an error is returned.

Examples:

```
alter view sale_detail_view rename to market;
```

6.4.3. Column and partition operations

6.4.3.1. Add partitions

This topic describes how to use a DDL statement to add partitions.

Syntax:

```
alter table table_name add [if not exists] partition partition_spec;
-- Add one partition at a time.
alter table table_name add [if not exists] partition partition_spec [PARTITION partition_spec PARTITION partition_spec...] ;
-- Add multiple partitions at a time.
```

Note

- If you do not specify IF NOT EXISTS and another partition with the same name exists, an error is returned.
- A MaxCompute table can contain a maximum of 60,000 partitions.
- To add a partition to a table that has multi-level partitions, you must specify all partition values.

Examples:

```
alter table sale_detail add if not exists partition (sale_date='201712', region='hangzhou');
-- Add a partition to store the sales records of the China (Hangzhou) region for December 2017.
alter table sale_detail add if not exists partition (sale_date='201712', region='shanghai');
-- Add a partition to store the sales records of the China (Shanghai) region for December 2017.
alter table sale_detail add if not exists partition(sale_date='20171011');
-- Specify only the sale_date partition. An error is returned.
alter table sale_detail add if not exists partition(region='shanghai');
-- Specify only the region partition. An error is returned.
alter table sale_detail add if not exists partition (sale_date='201712', region='hangzhou') partition
n (sale_date='201712', region='shanghai');
-- Add two partitions to store the sales records of the China (Hangzhou) and China (Shanghai) region
s for December 2017.
```

6.4.3.2. Delete a partition

This topic describes how to use DDL statements to delete a partition.

MaxCompute allows you to delete partitions that meet a specific filter condition. If you want to delete one or more partitions that meet a filter condition at a time, you can use an expression to specify the condition, use the condition to filter partitions, and then delete the partitions.

If you do not specify a filter condition, execute the following statements:

```
alter table table_name drop [if exists] PARTITION partition_spec;
-- Delete one partition at a time.
alter table table_name drop [if exists] PARTITION partition_spec,PARTITION partition_spec,[PARTITION
partition_spec....] ;
-- Delete multiple partitions at a time.
```

 **Note** Syntax for partition_spec:

```
partition_spec: (partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...
)
```

If you specify a filter condition, execute the following statement:

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_filtercondition;
-- Delete a partition based on the specified filter condition.
```

 **Note** Syntax for PARTITION partition_filtercondition:

```
PARTITION partition_filtercondition
: PARTITION (partition_col relational_operators partition_col_value)
| PARTITION (scalar(partition_col) relational_operators partition_col_value)
| PARTITION (partition_filtercondition1 AND|OR partition_filtercondition2)
| PARTITION (NOT partition_filtercondition)
| PARTITION (partition_filtercondition1)[,PARTITION (partition_filtercondition2), ...]
```

Parameters:

- **table_name**: the name of the partitioned table from which you want to delete a partition.
- **IF EXISTS**: If IF EXISTS is not specified and the partition does not exist, an error is returned.
- **partition_spec**: the name of the partition that you want to delete. The value is not case-sensitive. **partition_col** indicates the name of the partition key column, and **partition_col_value** indicates the value of the partition key column.
- **PARTITION partition_filtercondition**: the partition that you want to delete. The value is not case-sensitive.
 - **partition_col**: the name of the partition key column.
 - **relational_operators**: the relational operator.
 - **partition_col_value**: a value in the partition key column, which is a comparison value or regular expression. The data type of this value must be the same as that of the partition key column.
 - **scalar()**: the scalar function. This function generates a scalar based on the input value, processes the value in **partition_col**, and uses the relational operator specified by **relational_operators** to compare the processed value with **partition_col_value**.
 - The filter condition that is used to delete partitions supports the following logical operators: NOT, AND, and OR. You can use **PARTITION (NOT partition_filtercondition)** to obtain the complementary set of the condition. You can use **PARTITION (partition_filtercondition1 AND|OR partition_filtercondition2)** to obtain the condition that is used to match the partitions you want to delete.
 - Multiple **PARTITION partition_filtercondition** clauses are supported. If these clauses are separated by commas (,), OR is used for each clause to obtain the condition that is used to match the partitions you want to delete.

Limits:

- Each **PARTITION partition_filtercondition** clause can be used for only one partition key column.
- If you use an expression to specify **PARTITION partition_filtercondition**, only the built-in scalar function can be used for the expression.

Examples:

The filter condition is not specified.

```
-- Delete a partition from the sale_detail table. The partition stores the sales records of the China (Hangzhou) region in December 2017.
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date='201712',region='hangzhou');
-- Delete two partitions from the sale_detail table at a time. The partitions store the sales records of the China (Hangzhou) and China (Shanghai) regions in December 2017.
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date='201712',region='hangzhou'),PARTITION(sale_date='201712',region='shanghai');
```

The filter condition is specified.

```
-- Create a partitioned table.
CREATE TABLE IF NOT EXISTS sale_detail(
shop_name      STRING,
customer_id    STRING,
total_price    DOUBLE)
PARTITIONED BY (sale_date STRING);
-- Add partitions.
ALTER TABLE sale_detail ADD IF NOT EXISTS
PARTITION (sale_date= '201910')
PARTITION (sale_date= '201911')
PARTITION (sale_date= '201912')
PARTITION (sale_date= '202001')
PARTITION (sale_date= '202002')
PARTITION (sale_date= '202003')
PARTITION (sale_date= '202004')
PARTITION (sale_date= '202005')
PARTITION (sale_date= '202006')
PARTITION (sale_date= '202007');
-- Delete multiple partitions from the table at a time.
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date < '201911');
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date >= '202007');
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date LIKE '20191%');
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date IN ('202002','202004','202006'));
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date BETWEEN '202001' AND '202007');
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(substr(sale_date, 1, 4) = '2020');
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date < '201912' OR sale_date >= '202006');
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date > '201912' AND sale_date <= '202004');
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(NOT sale_date > '202004');
-- Delete partitions by using a condition. The condition is specified by expressions that have the OR relationship.
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date < '201911'), PARTITION(sale_date >= '202007');
-- Add partitions in other formats.
ALTER TABLE sale_detail ADD IF NOT EXISTS
PARTITION (sale_date= '2019-10-05');
PARTITION (sale_date= '2019-10-06')
PARTITION (sale_date= '2019-10-07');
-- Delete multiple partitions at a time and use regular expressions to match the partitions you want to delete.
ALTER TABLE sale_detail DROP IF EXISTS PARTITION(sale_date RLIKE '2019-\\d+-\\d+');
-- Create a table that has multi-level partitions.
CREATE TABLE IF NOT EXISTS region_sale_detail(
shop_name      STRING,
customer_id    STRING,
total_price    DOUBLE)
PARTITIONED BY (sale_date STRING , region STRING );
-- Add partitions.
ALTER TABLE region_sale_detail ADD IF NOT EXISTS
```

```

PARTITION (sale_date= '201910',region = 'shanghai')
PARTITION (sale_date= '201911',region = 'shanghai')
PARTITION (sale_date= '201912',region = 'shanghai')
PARTITION (sale_date= '202001',region = 'shanghai')
PARTITION (sale_date= '202002',region = 'shanghai')
PARTITION (sale_date= '201910',region = 'beijing')
PARTITION (sale_date= '201911',region = 'beijing')
PARTITION (sale_date= '201912',region = 'beijing')
PARTITION (sale_date= '202001',region = 'beijing')
PARTITION (sale_date= '202002',region = 'beijing');
-- Delete the partitions that meet the sale_date < '201911' and region = 'beijing' conditions from the table at a time.
ALTER TABLE region_sale_detail DROP IF EXISTS PARTITION(sale_date < '201911'),PARTITION(region = 'beijing');

```

If you execute the following statement, the following error is returned: `FAILED: ODPS-0130071:[1,82] Semantic analysis exception - invalid column reference region, partition expression must have one and only one column reference .`

```

-- An error is returned because PARTITION partition_filtercondition can be specified only for one partition key column.
ALTER TABLE region_sale_detail DROP IF EXISTS PARTITION(sale_date < '201911' AND region = 'beijing')
;

```

6.4.3.3. Add columns or comments

This topic describes how to use DDL statements to add columns or comments.

Syntax:

```

ALTER TABLE table_name ADD COLUMNS (col_name1 type1,col_name2 type2...) ;
-- Add columns.
ALTER TABLE table_name ADD COLUMNS (col_name1 type1 comment 'XXX',col_name2 type2 comment 'XXX');
-- Add both columns and comments.

```

Examples:

```

ALTER TABLE sale_detail ADD COLUMNS (customer_name STRING, education BIGINT);
-- Add two columns to the sale_detail table.
ALTER TABLE sale_detail ADD COLUMNS (customer_name STRING comment 'Customer', education BIGINT comment 'Education' );
-- Add two columns and their comments to the sale_detail table.

```

6.4.3.4. Change the name of a column

This topic describes how to use a DDL statement to change the name of a column.

Syntax:

```

ALTER TABLE table_name CHANGE COLUMN old_col_name RENAME TO new_col_name;

```

Note

- The column specified by `old_col_name` must exist in the table.
- The table cannot contain the column specified by `new_col_name`.

Examples:

```
ALTER TABLE sale_detail CHANGE COLUMN customer_name RENAME TO customer;
-- Change the name of a column in the sale_detail table.
```

6.4.3.5. Modify the comment of a column

This topic describes how to use a DDL statement to modify the comment of a column.

Syntax:

```
ALTER TABLE table_name CHANGE COLUMN col_name COMMENT 'comment_string';
```

Note

- The value of `comment_string` cannot exceed 1,024 bytes.
- The data type and position of a column cannot be changed.
- The column specified by `col_name` must exist.

Examples:

```
ALTER TABLE sale_detail CHANGE COLUMN customer COMMENT 'customer';
-- Modify the comment of a column in the sale_detail table.
```

6.4.3.6. Modify the name and comment of a column at the same time

This topic describes how to use a DDL statement to modify the name and comment of a column at the same time.

Syntax:

```
ALTER TABLE table_name CHANGE COLUMN old_col_name new_col_name column_type COMMENT 'column_comment';
```

Note

- The column specified by `old_col_name` must exist.
- The table cannot contain the column specified by `new_col_name`.
- The data type and position of a column cannot be changed.
- The value of `column_comment` cannot exceed 1,024 bytes.

Examples:

```
ALTER TABLE sale_detail CHANGE COLUMN customer customer_name string COMMENT 'Customer';
-- Modify the name and comment of a column in the sale_detail table.
```

6.4.3.7. Change LastDataModifiedTime of a non-partitioned table or a partition in a partitioned table.

MaxCompute SQL allows you to execute the TOUCH operation to modify the value of LastDataModifiedTime for a non-partitioned table or a partition in a partitioned table. This operation changes the value of LastDataModifiedTime to the current time. This topic describes how to use a DDL statement to change the value of LastDataModifiedTime for a non-partitioned table or a partition in a partitioned table.

Syntax:

```
ALTER TABLE table_name TOUCH;
-- Change the value of LastDataModifiedTime for a non-partitioned table.
ALTER TABLE table_name TOUCH PARTITION(partition_col='partition_col_value', ...);
-- Change the value of LastDataModifiedTime for a partition in a partitioned table.
```

Note

- If the table specified by table_name does not exist, an error is returned.
- If the partition specified by partition_col='partition_col_value' does not exist, an error is returned.
- This operation changes the value of LastDataModifiedTime for a non-partitioned table or a partition in a partitioned table. In this case, MaxCompute considers a change to the table data and recalculates the lifecycle.

Sample statements:

```
ALTER TABLE result_table TOUCH;
-- Change the value of LastDataModifiedTime for a non-partitioned table.
ALTER TABLE sale_detail TOUCH PARTITION(sale_date='201712');
-- Change the value of LastDataModifiedTime for a partition in a partitioned table.
```

6.4.3.8. Modify partition values

This topic describes how to use a DDL statement to modify partition values.

Syntax:

```
ALTER TABLE table_name PARTITION (partition_col1 = 'partition_col_value1', partition_col2 = 'partition_col_value2', ...)
RENAME TO PARTITION (partition_col1 = 'partition_col_newvalue1', partition_col2 = 'partition_col_newvalue2', ...) ;
```

Note

- This statement can modify only the values rather than the names of partition key columns.
- To modify the values of one or more partitions in a table that has multi-level partitions, you must specify the partition values at each level.
- If the table specified by table_name does not exist, an error is returned.

Examples:

```
ALTER TABLE sale_detail PARTITION (sale_date = '201712', region = 'hangzhou') RENAME TO PARTITION (s
ale_date = '201710', region = 'beijing');
-- Modify the partition values in the sale_detail table.
```

6.4.3.9. Merge partitions

MaxCompute SQL allows you to execute the `MERGE PARTITION` statement to merge multiple partitions of a partitioned table into one partition. This operation deletes the dimension information about the merged partitions and transfers data to a specific partition. This topic describes how to use a DDL statement to merge partitions.

Syntax:

```
ALTER TABLE <tableName> MERGE [IF EXISTS] PARTITION(<predicate>) [, PARTITION(<predicate2>) ...] OVE
RWRITE PARTITION(<fullPartitionSpec>) [PURGE];
```

Note

- If you do not specify IF EXISTS and the partition that you want to merge does not exist, an error is returned.
- If you specify IF EXISTS but no partitions meet the merge conditions, no new partitions are generated.
- If operations such as INSERT, RENAME, and DROP are performed at the same time to modify the source data when you execute the preceding statement, an error is returned even if you specified IF EXISTS.
- If the PURGE attribute is specified, you cannot back up or restore merged partitions by using metadata snapshots.

Limits and troubleshooting:

- Extreme storage is not supported.
- Tables that depend on the file sequence are not supported, such as tables in Xlib or ALGO.
- External tables and table shards are not supported. After the partitions of a clustered table are merged, the merged partitions do not have the clustering attributes.
- Hash operations are performed by a catalog server on tables to merge partitions. A capacity limit is imposed on merged partitions. A hard link in Apsara Distributed File System can have a maximum of seven copies.
- You can merge a maximum of 4,000 partitions at a time.
- The number of partitions that can wait on a catalog server for merging is 10 million.
- If an error that indicates the catalog server is busy is returned, you can try again later.
- If a hard link in Apsara Distributed File System is faulty, you can clear the recycle bin and try again.

Sample code:

```

odps@ jet_zwz>SHOW PARTITIONS intpstringstringstring;
ds=20181101/hh=00/mm=00
ds=20181101/hh=00/mm=10
ds=20181101/hh=10/mm=00
ds=20181101/hh=10/mm=10
OK
odps@ jet_zwz>DESC intpstringstringstring;
+-----+-----+-----+-----+
| value  | ds      | hh      | mm      |
+-----+-----+-----+-----+
| 1      | 20181101 | 00      | 00      |
| 1      | 20181101 | 00      | 10      |
| 1      | 20181101 | 10      | 00      |
| 1      | 20181101 | 10      | 10      |
+-----+-----+-----+-----+
-- Partitions and data of a partitioned table:
odps@ jet_zwz>ALTER TABLE intpstringstringstring MERGE PARTITION(hh='00') OVERWRITE PARTITION(ds='20181101', hh='00', mm='00');
ID = 20190404025755844g80qwa7a
OK
-- Merge all partitions that meet the hh='00' condition into the ds=20181101/hh=00/mm=00 partition.
odps@ jet_zwz>SHOW PARTITIONS intpstringstringstring;
ds=20181101/hh=00/mm=00
ds=20181101/hh=10/mm=00
ds=20181101/hh=10/mm=10
OK
-- Query the partitions of the table after the merge operation is performed.
odps@ jet_zwz>DESC intpstringstringstring;
+-----+-----+-----+-----+
| value  | ds      | hh      | mm      |
+-----+-----+-----+-----+
| 1      | 20181101 | 00      | 00      |
| 1      | 20181101 | 00      | 00      |
| 1      | 20181101 | 10      | 00      |
| 1      | 20181101 | 10      | 10      |
+-----+-----+-----+-----+
-- Data in two partitions that meet the specified condition is merged into the specified partition.

```

6.5. DML statements

6.5.1. INSERT

6.5.1.1. Update data in a table

This topic describes how to use an INSERT statement to update data in a table.

The `INSERT OVERWRITE` and `INSERT INTO` statements are used to write the data processing results of MaxCompute SQL to a destination table.

The `INSERT INTO` statement inserts data into a table or partition. The `INSERT INTO` statement cannot be used to insert data into clustered tables. If you want to insert a small amount of test data, you can use this statement with `VALUES`.

The `INSERT OVERWRITE` statement clears the original data in a table or partition and inserts data into the table or partition. The `INSERT OVERWRITE` statement cannot be used to insert data into specific columns.

Syntax:

```
INSERT OVERWRITE|INTO TABLE table_name [PARTITION (partcol1=val1, partcol2=val2 ...)] [(col1,col2 ..
.))
select_statement
FROM from_statement
[ZORDER BY zcol1 [, zcol2 ...]] ;
```

Note

- The INSERT syntax in MaxCompute is different from that in MySQL or Oracle. In MaxCompute, `INSERT OVERWRITE` or `INSERT INTO` must be followed by the keyword TABLE, instead of directly followed by the table_name parameter.
- To ensure data consistency during concurrent writes, MaxCompute uses the atomicity, consistency, isolation, durability (ACID) properties.
- The mappings between the source and destination tables are based on the column sequence in the SELECT clause. The mappings between the column names of the tables are not considered.
- If the destination table consists of static partitions and you want to insert data into a static partition, partition key columns cannot be included in the SELECT clause.

Parameters:

- table_name: the name of the table into which you want to insert data.
- PARTITION (partcol1=val1, partcol2=val2 ...): the name of the partition into which you want to insert data. The value must be a constant. It cannot be an expression, such as a function.
- [(col1,col2 ...)]: the names of the columns into which you want to insert data. This parameter is not supported in the `INSERT OVERWRITE` statement.
- select_statement: the SELECT clause used to query the data that you want to insert from the source table.
- from_statement: the FROM clause used to indicate the data source. For example, the value can be the name of the source table.
- [ZORDER BY zcol1 [, zcol2 ...]]: the columns that are used to sort data. If you write data to a table or partition, you can use this clause to place rows with similar data records in the columns specified in select_statement in adjacent positions. This improves filtering performance for queries and reduces storage costs.

The `ORDER BY x, y` clause sorts data records based on the ordering of x coming before y. The `ZORDER BY x, y` clause places rows that have similar x values in adjacent positions and rows that have similar y values in adjacent positions. If the filter condition of an SQL query statement includes sort columns, the ORDER BY clause filters and sorts data based on x, whereas the ZORDER BY clause filters and sorts data based on x or on both x and y. ZORDER BY increases the column store ratio.

Note

- The mappings between the source and destination tables are based on the column sequence in the SELECT clause. The mappings between the column names of the tables are not considered.
- If the destination table consists of static partitions and you want to insert data into a partition, partition key columns cannot be included in the SELECT clause.
- The ZORDER BY clause occupies a large number of resources to write data. As a result, data writes based on ZORDER BY require a longer time than data writes without ordering.
- If the destination table is a clustered table, ZORDER BY is not supported.
- ZORDER BY can be used with DISTRIBUTE BY. However, ZORDER BY cannot be used with ORDER BY, CLUSTER BY, or SORT BY.

Examples:

Calculate the sales of different regions listed in the `sale_detail` table. Then, insert the obtained data into the `sale_detail_insert` table.

```
-- Create a partitioned table named sale_detail, add partitions, and insert data into the table. You
do not need to define partition key columns as standard columns in the statement that is used to cre
ate the source table.
CREATE TABLE IF NOT EXISTS sale_detail
(
shop_name      string,
customer_id    string,
total_price    double
)
PARTITIONED BY (sale_date STRING,region STRING);
-- Add partitions to the source table.
ALTER TABLE sale_detail ADD PARTITION (sale_date='2013', region='china');
-- Insert data into the source table. INSERT INTO TABLE table_name can be abbreviated as INSERT INTO
table_name. However, INSERT OVERWRITE TABLE table_name has no abbreviation.
INSERT INTO sale_detail PARTITION (sale_date='2013', region='china') VALUES ('s1','c1',100.1),('s2',
'c2',100.2),('s3','c3',100.3);
-- Create a destination table named sale_detail_insert that has the same schema as the source table.
CREATE TABLE sale_detail_insert LIKE sale_detail;
-- Add partitions to the destination table.
ALTER TABLE sale_detail_insert ADD PARTITION (sale_date='2013', region='china');
-- Extract data from the sale_detail table and insert the data into the sale_detail_insert table.
-- Fields in the destination table do not need to be declared or rearranged.
-- If the destination table consists of static partitions and partition fields have been declared in
PARTITION(), you do not need to include these fields in the SELECT clause. You only need to search f
or the fields based on the sequence of standard columns in the destination table and sequentially ma
p these fields to those in the destination table. If the destination table consists of dynamic parti
tions, partition fields must be included in the SELECT clause. For more information, see Insert data
into dynamic partitions.
INSERT OVERWRITE TABLE sale_detail_insert PARTITION (sale_date='2013', region='china')
SELECT
shop_name,
customer_id,
total_price
FROM sale_detail
ZORDER BY customer_id, total_price;
```

Take note of the following points:

- The mappings between the source and destination tables are based on the column sequence in the SELECT clause. The mappings between the column names of the tables are not considered. Example:

```
INSERT OVERWRITE TABLE sale_detail_insert PARTITION (sale_date='2013', region='china')
SELECT customer_id, shop_name, total_price FROM sale_detail;
```

In the created `sale_detail_insert` table, the columns `shop_name` STRING, `customer_id` STRING, and `total_price` BIGINT are listed in sequence. If you insert data from the `sale_detail` table to the `sale_detail_insert` table, the data is inserted from the `customer_id`, `shop_name`, and `total_price` columns in sequence. In this case, data in `sale_detail.customer_id` is inserted into `sale_detail_insert.shop_name`, and data in `sale_detail.shop_name` is inserted into `sale_detail_insert.customer_id`.

- If you insert data into a partition, partition key columns cannot be included in the SELECT clause. If you execute the following statement, an error is returned because `sale_date` and `region` are partition key columns. These columns cannot be included in the INSERT statement that is used to insert table data into a static partition.

```
INSERT OVERWRITE TABLE sale_detail_insert PARTITION (sale_date='2013', region='china')
SELECT shop_name, customer_id, total_price, sale_date, region FROM sale_detail;
```

- The value of PARTITION must be a constant and cannot be an expression. The following example shows invalid settings:

```
INSERT OVERWRITE TABLE sale_detail_insert PARTITION (sale_date=datepart('2016-09-18 01:10:00', 'yy
yy') , region='china')
SELECT shop_name, customer_id, total_price FROM sale_detail;
```

Precautions

If you want to update table data to a dynamic partition, take note of the following points:

- If you execute the `INSERT INTO PARTITION` statement and the specified partition does not exist, the system automatically creates this partition.
- If multiple jobs that use the `INSERT INTO PARTITION` statement concurrently run and the specified partitions do not exist, the system attempts to create these partitions. However, only one partition can be created.
- If you cannot control the concurrency of the jobs that use the `INSERT INTO PARTITION` statement, we recommend that you use the `ALTER TABLE` statement to create partitions in advance.

6.5.1.2. Insert data into multiple objects

This topic describes how to use an INSERT statement to insert data to multiple objects.

MaxCompute SQL allows you to insert data into different result tables or partitions by using one SQL statement.

Syntax:

```
FROM from_statement
INSERT OVERWRITE | INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement1 [FROM from_statement]
[INSERT OVERWRITE | INTO TABLE tablename2 [PARTITION (partcol1=val3, partcol2=val4 ...)]
select_statement2 [FROM from_statement]];
```

Note

- In most cases, an SQL statement supports up to 256 outputs. If the number of outputs exceeds 256, a syntax error is returned.
- In a MULTIINSERT statement, non-partitioned tables and destination partitions in a partitioned table must be unique.
- The `INSERT OVERWRITE` and `INSERT INTO` operations cannot be simultaneously performed on different partitions of a table. If these operations are simultaneously performed on the partitions, an error is returned.
- PARTITION (partcol1=val1, partcol2=val2 ...): The parameter values can be only constants rather than expressions, such as functions.

Examples:

```
-- Create a destination table named sale_detail_multi.
create table sale_detail_multi like sale_detail;
-- Insert data from sale_detail into sale_detail_multi.
set odps.sql.allow.fullscan=true; -- Enable the full table scan, which is valid only for this session.
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2011', region='china' )
select shop_name, customer_id, total_price ;
-- An error is returned because the same partition appears more than once.
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price;
-- An error is returned because the INSERT OVERWRITE and INSERT INTO operations are simultaneously performed on different partitions of a partitioned table.
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert into table sale_detail_multi partition (sale_date='2011', region='china' )
select shop_name, customer_id, total_price;
```

6.5.1.3. Insert data into dynamic partitions

This topic describes how to use an INSERT statement to insert data into dynamic partitions.

You can use the INSERT OVERWRITE statement to insert data into the following partitions of a partitioned table:

- **Static partitions:** You must specify partition values in the INSERT statement to insert data into the specified partitions.
- **Dynamic partitions:** You need only to specify the names of partition key columns in the INSERT statement. The values of the partition key columns are provided in the SELECT clause. The system automatically inserts data into the required partitions based on the values.

Syntax:

```
INSERT OVERWRITE|INTO TABLE tablename PARTITION (partcol1, partcol2 ...)
select_statement FROM from_statement;
```

Note

- A maximum of 10,000 dynamic partitions can be generated by using the INSERT INTO statement. A maximum of 60,000 dynamic partitions can be generated by using the INSERT OVERWRITE statement.
- An SQL statement that is used to insert data into dynamic partitions in a distributed environment can generate a maximum of 512 dynamic partitions in a single process. If the number of dynamic partitions exceeds this limit, an exception occurs.
- The dynamically generated partition values cannot be NULL or contain special characters. If a value is NULL or contains special characters, an exception occurs.

```
FAILED: ODPS-0123031:Partition exception - invalid dynamic partition value: province=xxx
```

- If the destination table has multi-level partitions, you can specify some partitions as static partitions in an INSERT statement. However, the static partitions must be high-level partitions.
- If the destination table is a hash-clustered table, dynamic partitions are not supported.
- If you insert data into dynamic partitions, the mappings between the columns in select_statement and dynamic partitions in the destination table are determined by the column sequence, not by the column names. If the sequence of columns in the source table is different from that in the destination table, we recommend that you specify the columns in select_statement based on the column sequence in the destination table.

Examples:

Insert data from a source table into a destination table. You can obtain the partitions generated based on the region field only after the required statement is executed.

```
-- Create a destination table named total_revenues.
create table total_revenues (revenue bigint) partitioned by (region string);
-- Insert the data from sale_detail into total_revenues.
insert overwrite table total_revenues partition(region)
select total_price as revenue, region from sale_detail;
```

Insert data from a source table into a destination table. If the destination table has multi-level partitions, level-1 partition sale_date must be specified.

```
insert overwrite table sale_detail_dypart partition (sale_date='2013', region)
select shop_name,customer_id,total_price,region from sale_detail;
```

To insert data into dynamic partitions, the dynamic partition key columns must be specified in the SELECT clause. Otherwise, the execution fails.

```
insert overwrite table sale_detail_dypart partition (sale_date='2013', region)
select shop_name,customer_id,total_price from sale_detail;
```

If you specify only low-level subpartitions when you insert data into dynamic partitions, you may fail to insert data into high-level partitions.

```
insert overwrite table sales partition (region='china', sale_date)
select shop_name,customer_id,total_price,sale_date from sale_detail;
```

If the data types of values in partition key columns are inconsistent with those in the SELECT clause, an error is returned when data is inserted into dynamic partitions in MaxCompute V1.0. MaxCompute V2.0 supports the implicit conversions of data types.

```
-- Create a destination table named parttable.
create table parttable(a int, b double) partitioned by (p string);
-- Insert the data from source table src into destination table parttable.
insert into parttable partition(p) select key, value, current_timestmap() from src;
-- Query data in parttable.
select * from parttable;
```

6.5.1.4. VALUES

This topic describes how to use the INSERT ...VALUES statement to insert data into a table that contains small amounts of data.

Syntax:

```
INSERT INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2,...)] [(colname1,colname2,...)]
[VALUES (col1_value,col2_value,...), (col1_value,col2_value,...),...]
```

Parameters:

- **tablename**: the name of the table into which you want to insert data. The table must be an existing table.
- **[PARTITION (partcol1=val1, partcol2=val2,...)]**: the information about partitions. If you want to insert data into a partitioned table, you must specify this parameter.
- **[(colname1,colname2,...)]**: the names of the columns in the destination table.
- **col_value**: the value in the specified column in the destination table. Separate multiple values with commas (,). Each value in the column must be a constant. If no values are specified, the default value NULL is used.

Note

- You can use only the `INSERT INTO` statement rather than the `INSERT OVERWRITE` statement to insert data into specific columns.
- Functions cannot be used to construct constants for some complex data types, such as ARRAY, in VALUES. However, you can use the following statement to pass the values of the ARRAY type to VALUES:

```
INSERT INTO TABLE srcp (p='abc') SELECT 'a', ARRAY('1', '2', '3');
```

- To pass the values of the DATETIME or TIMESTAMP type, you must specify the data type in VALUES.

```
INSERT INTO TABLE srcp (p='abc') VALUES (datetime'2017-11-11 00:00:00',timestamp'2017-11-11 00:00:00.123456789');
```

Example:

Insert data into a specific partition.

```
-- Delete the srcp table that already exists in the system.
DROP TABLE IF EXISTS srcp;
-- Create a partitioned table named srcp.
CREATE TABLE IF NOT EXISTS srcp (key string,value bigint) PARTITIONED BY (p string);
-- Insert data into the abc partition of the srcp table.
INSERT INTO TABLE srcp PARTITION (p='abc') VALUES ('a',1), ('b',2), ('c',3);
-- Query data from the srcp table.
SELECT * FROM srcp WHERE p='abc';
```

The following information is returned:

```

+-----+-----+-----+
| key    | value  | p      |
+-----+-----+-----+
| a      | 1      | abc    |
| b      | 2      | abc    |
| c      | 3      | abc    |
+-----+-----+-----+

```

Insert data into any partition.

```

-- Delete the srcp table that already exists in the system.
DROP TABLE IF EXISTS srcp;
-- Create a partitioned table named srcp.
CREATE TABLE IF NOT EXISTS srcp (key string,value bigint) PARTITIONED BY (p string);
-- Insert data into a partition of the srcp table.
INSERT INTO TABLE srcp PARTITION (p) (key,p) VALUES ('d','20170101'),('e','20170101'),('f','20170101'
);
-- Query data from the srcp table.
SELECT * FROM srcp WHERE p='20170101';

```

The following information is returned:

```

+-----+-----+-----+
| key    | value  | p      |
+-----+-----+-----+
| d      | NULL   | 20170101 |
| e      | NULL   | 20170101 |
| f      | NULL   | 20170101 |
+-----+-----+-----+

```

Use complex data types to construct constants and execute an `INSERT` statement to import data.

```

-- Create a partitioned table named srcp.
create table if not exists srcp (key string,value array<int>) partitioned by (p string);
-- Add a partition to the srcp table.
alter table srcp add if not exists partition (p='abc');
-- Insert data into the abc partition of the srcp table.
insert into table srcp partition (p='abc') select 'a', array(1, 2, 3);
-- Query data from the srcp table.
select * from srcp where p='abc';

```

The following information is returned:

```

+-----+-----+-----+
| key    | value  | p      |
+-----+-----+-----+
| a      | [1,2,3] | abc    |
+-----+-----+-----+

```

Use the `INSERT...VALUES` statement to write data of the DATETIME or TIMESTAMP type.

```
-- Create a partitioned table named srcp.
create table if not exists srcp (key string, value timestamp) partitioned by (p string);
-- Add a partition to the srcp table.
alter table srcp add if not exists partition (p='abc');
-- Insert data into the abc partition of the srcp table.
insert into table srcp partition (p='abc') values (datetime'2017-11-11 00:00:00',timestamp'2017-11-11 00:00:00.123456789');
-- Query data from the srcp table.
select * from srcp where p='abc';
```

The following information is returned:

```
+-----+-----+-----+
| key      | value      | p      |
+-----+-----+-----+
| 2017-11-11 00:00:00 | 2017-11-11 00:00:00.123 | abc      |
+-----+-----+-----+
```

Features of VALUES TABLE

When you use INSERT...VALUES, the values after VALUES must be constants. If you want to perform simple computing operations on the inserted data, we recommended that you use VALUES TABLE of MaxCompute.

Syntax:

```
values (<col1_value>,<col2_value>,...), (<col1_value>,<col2_value>,...),<table_name> (<col1_name> ,<col2_name>,...)...
```

You can use VALUES TABLE in the following scenarios:

- If no physical tables are available, create a table that contains multiple rows of data and perform computing operations on the table.

For example, `VALUES (...), (...) t(a, b)` defines that a table named t contains the a and b columns. The data type of the a column is STRING and that of the b column is BIGINT. The data type of a column must be derived from the VALUES list.

```
-- Delete the srcp table that already exists in the system.
DROP TABLE IF EXISTS srcp;
-- Create a partitioned table named srcp.
CREATE TABLE IF NOT EXISTS srcp (key string,value bigint) PARTITIONED BY (p string);
-- Insert data into the srcp table.
INSERT INTO TABLE srcp PARTITION (p) SELECT concat(a,b), length(a)+length(b), '20170102' FROM VALUES ('d',4), ('e',5), ('f',6) t(a,b);
-- Query data from the srcp table.
SELECT * FROM srcp WHERE p='20170102';
```

The following information is returned:

```
+-----+-----+-----+
| key      | value      | p      |
+-----+-----+-----+
| d4       | 2          | 20170102 |
| e5       | 2          | 20170102 |
| f6       | 2          | 20170102 |
+-----+-----+-----+
```

- Use VALUES TABLE rather than the combination of `SELECT * FROM` and `UNION ALL` to create a constant-

type table.

```
SELECT 1 c UNION ALL SELECT 2 c;
-- The preceding statement is equivalent to the following statement:
SELECT * FROM VALUES (1), (2) t(c);
```

The following information is returned:

```
+-----+
| c      |
+-----+
| 1      |
| 2      |
+-----+
```

- **Special forms of VALUES TABLE.** If the expressions of the SELECT statement do not contain upstream table data, you can execute the SELECT statement without the FROM clause. In this case, the underlying implementation is to select data from an anonymous VALUES table that contains only one row and no columns. This way, you can test user-defined functions (UDFs) or other functions without the need to manually create DUAL tables.

```
-- Create a partitioned table named srcp.
create table if not exists srcp (key string,value bigint) partitioned by (p string);
-- Insert data into the srcp table.
insert into table srcp partition (p) select abs(-1), length('abc'), getdate();
-- Query data from the srcp table.
select * from srcp;
```

The following information is returned:

```
+-----+-----+-----+
| key      | value  | p          |
+-----+-----+-----+
| 1        | 3      | 2020-11-25 18:39:48 |
+-----+-----+-----+
```

Support for expressions that do not contain constants

Constant expressions are supported in VALUES expressions. Expressions that do not contain constants are also supported in VALUES expressions. Syntax:

```
select * from values (udf(1)),(to_date('20190101', 'yyyymmdd')), (getdate()) t(d);
```

6.5.2. SELECT

6.5.2.1. SELECT operations

This topic describes how to use SELECT statements for SELECT operations.

Syntax:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[ORDER BY order_condition]
[DISTRIBUTE BY distribute_condition [SORT BY sort_condition] ]
[LIMIT number];
```

Limits

- A SELECT statement can return a maximum of 10,000 rows of results. However, no limits are imposed when the SELECT statement is used as a clause. If the SELECT statement is used as a clause, the clause returns all results in response to the query from the upper layer.
- You are not allowed to perform full table scans on partitioned tables by using SELECT statements. To perform a full table scan on a partitioned table, add `set odps.sql.allow.fullscan=true;` before the SQL statement that is used for the full table scan. Then, commit and run the added command with the SQL statement. To enable the full table scan for the entire project, run the following command:

```
setproject odps.sql.allow.fullscan=true;
```

Column expression (select_expr)

Each select_expr expression indicates a column from which you want to read data. You can use the following select_expr expressions to read data from a table.

- Specify the names of the columns from which you want to read data.

```
select shop_name from sale_detail;
```

- Use an asterisk (*) to represent all columns.

```
select * from sale_detail;
-- You can also use a WHERE clause to specify filter conditions.
select * from sale_detail where shop_name like 'hang%';
```

- Use a regular expression.
- Use DISTINCT before the name of a column to filter out duplicate values from the column and return only distinct values. If you use ALL before the name of a column, all values including the duplicate values in the column are returned. If DISTINCT is not used, the default value ALL is used.

```
-- Query data from the region column in the sale_detail table and return only distinct values.
select distinct region from sale_detail;
```

The following result is returned:

```
+-----+
| region |
+-----+
| shanghai |
+-----+
```

```
-- If you use DISTINCT in a SELECT statement where multiple columns are specified, DISTINCT applies to all the columns you specified, instead of a single column.
select distinct region, sale_date from sale_detail;
```

The following result is returned:

```
+-----+-----+
| region | sale_date |
+-----+-----+
| shanghai | 20191110 |
+-----+-----+
```

Information about a queried destination table (table_reference)

table_reference specifies the table that you want to query. Nested subqueries are also supported in table_reference. Example:

```
select * from (select region from sale_detail) t where region = 'shanghai';
```

WHERE

The following table describes the operators that are used as filter conditions supported by the WHERE clause.

Operator	Description
>, <, =, ≥, ≤, and <>	The relational operators.
LIKE and RLIKE	The Source and Pattern parameters of LIKE and RLIKE must be of the STRING type.
IN and NOT IN	If a subquery is added after the IN or NOT IN operator, the values in only one column can be returned for the subquery and the number of return values cannot exceed 1,000.
BETWEEN...AND	The operator that specifies the query range.

In the WHERE clause of a SELECT statement, you can specify the partitions that you want to scan in a table to avoid a full table scan.

```
SELECT sale_detail.*
FROM sale_detail
WHERE sale_detail.sale_date >= '2008'
AND sale_detail.sale_date <= '2014';
```

User-defined functions (UDFs) support partition pruning. UDFs are executed as small jobs and then replaced with the execution results. You can use one of the following methods to implement UDF-based partition pruning:

- Add an annotation to the UDF class when you write a UDF.

```
@com.aliyun.odps.udf.annotation.UdfProperty(isDeterministic=true)
```

 **Notice** com.aliyun.odps.udf.annotation.UdfProperty defines that the version of referenced odps-sdk-udf must be 0.30.x or later in odps-sdk-udf.jar.

- Add the `set odps.sql.udf.ppr.deterministic = true;` flag before SQL statements. Then, all UDFs in the SQL statements are considered deterministic.

 **Note** This method backfills partitions with execution results. A maximum of 1,000 partitions can be backfilled with execution results. If you add an annotation to a UDF class, an error may be returned, which indicates that more than 1,000 partitions are backfilled with execution results. If you want to ignore the error, add the `set odps.sql.udf.ppr.to.subquery = false;` flag to globally disable this feature. After this feature is disabled, UDF-based partition pruning no longer takes effect.

The WHERE clause in an SQL statement can include the BETWEEN...AND operator. Example:

```
SELECT sale_detail.*
FROM sale_detail
WHERE sale_detail.sale_date BETWEEN '2008' AND '2014';
```

GROUP BY

In most cases, the GROUP BY clause is used with aggregate functions. If a SELECT statement contains aggregate functions, the following rules apply:

- During the parsing of SQL statements, the GROUP BY clause precedes a SELECT operation. Therefore, the value of the GROUP BY clause must be the names of the columns in the input table for the SELECT operation. The value can also be an expression composed of the columns in the input table. The value cannot be the aliases of the output columns from the SELECT operation.
- The value of the GROUP BY clause may be both a column name or an expression composed of input table columns and the alias of an output column from the SELECT operation. In this case, the column name or expression is used as the value of the GROUP BY clause.
- If the `set hive.groupby.position.alias=true;` flag is added before a SELECT statement, the integer constants in GROUP BY are considered column numbers in the SELECT operation.

```
set hive.groupby.position.alias=true;
-- Run this command with the next SELECT statement.
select region, sum(total_price) from sale_detail group by 1;
-- 1 indicates the region column, which is the first column read by the SELECT statement. This statement groups the table data based on the values in the region column and returns distinct region values and total sales of each group.
```

Examples:

```
-- The statement uses the name of the region column in the input table as the value of the GROUP BY clause and groups the table data based on the values in the region column.
select region from sale_detail group by region;
-- The statement groups the table data based on the values in the region column and returns the total sales of each group.
select sum(total_price) from sale_detail group by region;
-- The statement groups the table data based on the values in the region column and returns distinct values and total sales of each group.
select region, sum(total_price) from sale_detail group by region;
-- An error is returned because the alias of the column in the SELECT operation is used as the value of the GROUP BY clause.
select region as r from sale_detail group by r;
-- A complete expression of the column is required.
select 2 + total_price as r from sale_detail group by 2 + total_price;
-- An error is returned because all the columns that do not use aggregate functions in the SELECT operation must be included in the GROUP BY clause.
select region, total_price from sale_detail group by region;
-- The statement can be successfully executed because all the columns that do not use aggregate functions are included in the GROUP BY clause.
select region, total_price from sale_detail group by region, total_price;
```

ORDER BY|DISTRIBUTE BY|SORT BY

- ORDER BY: This clause is used to sort all data records based on the specified columns.
 - To sort data records in descending order, use the desc keyword. By default, data records are sorted in ascending order.
 - When you use ORDER BY to sort data records, NULL is considered the smallest value. This rule is consistent with MySQL, but is different from Oracle.
 - The columns in the ORDER BY clause must be the aliases of the columns in the SELECT operation. If you want to query a column but the column alias is not specified in the SELECT operation, the column name is used as the column alias.

- If the `set hive.orderby.position.alias=true;` flag is added before SQL statements, integer constants in the ORDER BY clause are considered column numbers in a SELECT operation.

```
-- Set the flag.
set hive.orderby.position.alias=true;
-- Create a table named src.
create table src(key BIGINT,value BIGINT);
-- Query the src table and sort the records that are returned in ascending order by value.
SELECT * FROM src ORDER BY 2 limit 100;
The preceding statement is equivalent to the following statement:
SELECT * FROM src ORDER BY value limit 100;
```

- You can use the OFFSET clause with the ORDER BY LIMIT clause to skip the number of rows specified by OFFSET.

```
-- Sort the rows of the src table in ascending order by key, and return row 11 to row 30. OFFSET
10 indicates that the first 10 rows are skipped, and LIMIT 20 indicates that a maximum of 20 row
s can be returned.
SELECT * FROM src ORDER BY key LIMIT 20 OFFSET 10;
```

Examples:

```
-- Query the sale_detail table and sort the first 100 records that are returned in ascending order
by region.
SELECT * FROM sale_detail ORDER BY region LIMIT 100;
-- By default, the LIMIT clause is used with the ORDER BY clause. If only the ORDER BY clause is u
sed, an error is returned.
SELECT * FROM sale_detail ORDER BY region;
-- ORDER BY is followed by a column alias.
SELECT region AS r FROM sale_detail ORDER BY region LIMIT 100;
SELECT region AS r FROM sale_detail ORDER BY r LIMIT 100;
```

- **DISTRIBUTE BY:** This clause is used to shard data based on hash values of specific columns. You must specify the alias of an output column from the SELECT operation as the value of the DISTRIBUTE BY clause.

Examples:

```
-- The statement queries values in the region column of the sale_detail table and shards data base
d on the hash values of the region column.
SELECT region FROM sale_detail DISTRIBUTE BY region;
-- The statement can be successfully executed because the column name is used as the alias.
SELECT region AS r FROM sale_detail DISTRIBUTE BY region;
The preceding statement is equivalent to the following statement:
SELECT region AS r FROM sale_detail DISTRIBUTE BY r;
```

- **SORT BY:** This clause is used to sort specific data records. You can add a keyword, such as ASC or DESC, to sort specific data records. If ASC is used, data records are sorted in ascending order. If DESC is used, data records are sorted in descending order. If SORT BY is not followed by a keyword, data records are sorted in ascending order by default.

- If the SORT BY clause is preceded by the DISTRIBUTE BY clause, the SORT BY clause sorts specific results of the DISTRIBUTE BY clause.

The DISTRIBUTE BY clause determines how to distribute the output values of the map stages to reducers. To prevent duplicate output values from being distributed to different reducers or to process the same group of data together, you can use the DISTRIBUTE BY clause. This ensures that the same group of data is distributed to the same reducer. Then, use SORT BY to sort the group of data. Example:

```
-- The statement queries values in the region column of the sale_detail table, shards data based
on the hash values of the region column, and then sorts the sharded data.
SELECT region FROM sale_detail DISTRIBUTE BY region SORT BY region;
```

- If the SORT BY clause is not preceded by the DISTRIBUTE BY clause, the SORT BY clause sorts the data of each reducer. This process sorts specific data records. This ensures that the output values of each reducer are sorted and increases the data compression ratio. If data is filtered during data reading, the amount of data read from disks is reduced, which makes subsequent global sorting more efficient. Example:

```
SELECT region FROM sale_detail SORT BY region;
```

Note

- The value of ORDER BY, DISTRIBUTE BY, or SORT BY must be the alias of an output column from the SELECT operation. The column alias can be Chinese.
- During the parsing of MaxCompute SQL statements, the ORDER BY, DISTRIBUTE BY, or SORT BY clause is executed after the SELECT operation. Therefore, the value of ORDER BY, DISTRIBUTE BY, or SORT BY must be the alias of an output column from the SELECT operation.
- The ORDER BY clause cannot be used with the DISTRIBUTE BY or SORT BY clause. The GROUP BY clause cannot be used with the DISTRIBUTE BY or SORT BY clause.

LIMIT

The number in the LIMIT clause is a constant that limits the number of rows to return.

- Remove the limit on the simultaneous execution of the ORDER BY and LIMIT clauses

The ORDER BY clause needs to sort all data of a single node. By default, the ORDER BY clause is used with the LIMIT clause to prevent a single node from processing large amounts of data. You can remove the limit on the simultaneous execution of the ORDER BY and LIMIT clauses for a project or session.

- To remove the limit for a project, run the `setproject odps.sql.validate.orderby.limit=false;` command.
- To remove the limit for a session, run the `setproject odps.sql.validate.orderby.limit=false;` command with the SQL statement.

 **Note** If a single node has large amounts of data to sort after the limit is removed, more resources and time are required.

- Limits on the number of rows to return

If you use a SELECT statement without the LIMIT clause or the number in the LIMIT clause exceeds the specified upper limit, you can view only the rows within the upper limit.

The upper limit on the number of rows to return varies based on projects. You can use one of the following methods to configure the upper limit:

- If project protection is disabled, `use_instance_tunnel` is set to `true`, and `instance_tunnel_max_record` is not specified, no limits are imposed on the number of rows to return. For more information about project protection, see [Data protection](#). For more information about the configurations of `use_instance_tunnel=true` and `instance_tunnel_max_record`, see [Tunnel operations](#).
- If project protection is enabled, the number of rows to return is limited by `READ_TABLE_MAX_ROW`. The maximum value of this parameter is 10000.

Note You can run the `show SecurityConfiguration;` command to view the configuration of ProjectProtection. If ProjectProtection is set to `true`, you can determine whether to disable project protection. You can run the `set ProjectProtection=false;` command to disable project protection. ProjectProtection is disabled by default. For more information about project protection, see [Data protection](#).

6.5.2.2. Subquery

This topic describes how to use a SELECT statement to perform subqueries.

A common SELECT statement reads data from multiple tables, such as, `select column_1, column_2 ... from table_name`. The query object can be another SELECT operation, which is a subquery.

Syntax:

```
select * from (select shop_name from sale_detail) a;
```

Notice A subquery must have an alias.

Examples:

```
create table shop as select * from sale_detail;
select a.shop_name, a.customer_id, a.total_price from
(select * from shop) a join sale_detail on a.shop_name = sale_detail.shop_name;
```

Note In a FROM clause, a subquery can be used as a table, which supports a JOIN operation with other tables or subqueries.

6.5.2.3. UNION ALL

This topic describes how to use a SELECT statement to perform UNION ALL operations.

Syntax:

```
select_statement union all select_statement
```

Note The UNION ALL clause is used to combine two or more datasets returned from a SELECT operation into one dataset. If duplicate rows exist in the results, all rows that meet the condition are returned, with duplicate rows retained.

MaxCompute SQL does not support UNION ALL between two top-level query statements. To combine the query results of the two statements, you must rewrite the UNION ALL clause as a subquery.

Syntax before rewriting:

```
select * from sale_detail where region = 'hangzhou'
union all
select * from sale_detail where region = 'shanghai';
```

Syntax after rewriting:

```
select * from (
select * from sale_detail where region = 'hangzhou' union all
select * from sale_detail where region = 'shanghai') t;
```

The syntax that uses a pair of parentheses to specify the priority of UNION ALL is supported.

Examples:

```
SELECT * FROM src UNION ALL (SELECT * FROM src2 UNION ALL SELECT * FROM src3);
-- Execute the UNION ALL clause for the src2 and src3 tables. Then, execute the UNION ALL clause for
the src table and the result of the first UNION ALL operation.
```

Notice

- For a UNION ALL operation, all subqueries must have the same number of columns, column names, and column types. If the column names are inconsistent, use column aliases.
- In most cases, MaxCompute allows a UNION ALL operation for a maximum of 256 subqueries. If the limit is exceeded, a syntax error is returned.

6.5.2.4. JOIN

This topic describes how to use SELECT statements to perform JOIN operations.

MaxCompute supports multiple JOIN operations in an SQL statement. MaxCompute does not support CROSS JOIN. A CROSS JOIN operation joins two tables without the need to specify conditions in the ON clause.

Syntax:

```
join_table:
    table_reference JOIN table_factor [join_condition]
    | table_reference {LEFT OUTER|RIGHT OUTER|FULL OUTER|INNER|NATURAL} JOIN table_reference join_condition
table_reference:
    table_factor
    | join_table
table_factor:
    tbl_name [alias]
    | table_subquery alias
    | ( table_references )
join_condition:
    ON equality_expression ( AND equality_expression )
```

 **Note** equality_expression indicates an equality expression.

Types of JOIN operations:

- LEFT OUTER JOIN: shortened as LEFT JOIN. It returns all rows in the left table, including the rows that do not match any rows in the right table.

```
SELECT a.shop_name AS ashop, b.shop_name AS bshop FROM shop a
LEFT OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
-- Both the shop and sale_detail tables have the shop_name column. You must use aliases to distinguish the columns in the SELECT operation.
```

Note If the values in some rows of the right table are duplicate, we recommend that you do not consecutively use LEFT JOIN. If you consecutively use LEFT JOIN, data bloat may occur and interrupt your jobs.

- **RIGHT OUTER JOIN:** shortened as **RIGHT JOIN**. It returns all rows in the right table, including rows that do not match any rows in the left table.

```
SELECT a.shop_name AS ashop, b.shop_name AS bshop FROM shop a
RIGHT OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

- **FULL OUTER JOIN:** shortened as **FULL JOIN**. It returns all rows in both the left and right tables.

```
SELECT a.shop_name AS ashop, b.shop_name AS bshop FROM shop a
FULL OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

- **INNER JOIN:** The **INNER** keyword can be omitted. **INNER JOIN** only returns rows in which a match between the two tables exists.

```
SELECT a.shop_name FROM shop a INNER JOIN sale_detail b ON a.shop_name=b.shop_name;
SELECT a.shop_name FROM shop a JOIN sale_detail b ON a.shop_name=b.shop_name;
```

- **NATURAL JOIN:** In a **NATURAL JOIN** operation, the conditions used to join two tables are automatically determined based on the common fields in the two tables. MaxCompute supports **OUTER NATURAL JOIN**. You can use the **USING** clause so that the **JOIN** operation returns common fields only once.

```
SELECT * FROM src NATURAL JOIN src2;
-- Both the src and src2 tables include the key1 and key2 fields. In this case, the preceding statement is equivalent to the following statement:
SELECT src.key1 AS key1, src.key2 AS key2, src.a1, src.a2, src2.b1, src2.b2 FROM src INNER JOIN src2 ON src.key1 = src2.key1 AND src.key2 = src2.key2;
```

- **Implicit JOIN operation:** A **JOIN** operation that is performed without the need to specify the **JOIN** keyword.

```
SELECT * FROM table1, table2 WHERE table1.id = table2.id;
-- The preceding statement is equivalent to the following statement:
SELECT * FROM table1 JOIN table2 ON table1.id = table2.id;
```

Join conditions: You must use equi-joins and combine conditions by using **AND**. You can use non-equi joins or combine multiple conditions by using **OR** in a **MAPJOIN** operation.

```
-- MaxCompute supports multiple JOIN operations in an SQL statement.
SELECT a. * FROM shop a FULL OUTER JOIN sale_detail b ON a.shop_name=b.shop_name
      FULL OUTER JOIN sale_detail c ON a.shop_name=c.shop_name;
-- MaxCompute does not support non-equi joins and returns an error.
SELECT a. * FROM shop a JOIN sale_detail b ON a.shop_name != b.shop_name;
```

You can use parentheses **()** to specify the priority of **JOIN** operations. The **JOIN** operation enclosed in parentheses **()** has a high priority.

```
SELECT * FROM src JOIN (src2 JOIN src3 on xxx) ON yyy;
-- In the preceding statement, src2 JOIN src3 is first executed. Then, the JOIN operation is performed for the src table and the result of the first JOIN operation.
```

Examples:

The `test_table_a` and `test_table_b` tables are available. You want to query the two tables for the rows whose values in the `origin` column are equal to those in the `id` column and whose data timestamp is greater than 20180101. If you use `LEFT JOIN`, all rows in the left table `test_table_a` are returned.

```
SELECT s.id
       ,s.name
       ,s.origin
       ,d.value
FROM   (SELECT * FROM test_table_a WHERE ds > "20180101" ) s
LEFT JOIN (SELECT * FROM test_table_b WHERE ds > "20180101") d
ON s.origin = d.id;
```

6.5.2.5. MAPJOIN hint

This topic describes how to specify `MAPJOIN` in a `SELECT` statement to join a large table with one or more small tables.

Background information

You can explicitly specify `MAPJOIN` in a `SELECT` statement to join a large table with one or more small tables. `MAPJOIN` speeds up your query.

A `JOIN` operation contains three stages: map, shuffle, and reduce. In most cases, tables are joined in the reduce stage.

`MAPJOIN` joins tables in the map stage instead of the reduce stage. This reduces data transmission time and system resource consumption and optimizes jobs.

In the map stage, `MAPJOIN` loads all data in the specified tables into the memory of the program that executes the `JOIN` operation. The tables specified for `MAPJOIN` must be small tables, and the total memory occupied by the table data cannot exceed 512 MB.

When a large table is joined with one or more small tables, `MAPJOIN` loads all data in the specified small tables into the memory of the program that executes the `JOIN` operation. This way, tables are connected in the map stage to accelerate the execution of the `JOIN` operation.

Usage

To use `MAPJOIN`, you must specify the hint `/*+ MAPJOIN(table) */` in a `SELECT` statement.

For example, `sale_detail` is a large table whose alias is `b`, and `shop` is a small table whose alias is `a`.

- The following statement is used to perform a common `JOIN` operation:

```
SELECT a.shop_name,
       b.customer_id,
       b.total_price
FROM   shop a JOIN sale_detail b
ON a.shop_name = b.shop_name;
```

- The following statement is used to perform a `JOIN` operation with `MAPJOIN` specified:

```
SELECT /*+ MAPJOIN(a) */
       a.shop_name,
       b.customer_id,
       b.total_price
FROM   shop a JOIN sale_detail b
ON a.shop_name = b.shop_name;
```

Limits

- When you use MAPJOIN and reference a small table or a subquery, you must reference the alias of the table or subquery.
- MAPJOIN supports small tables in subqueries.
- The left table in a LEFT OUTER JOIN operation must be a large table.
- The right table in a RIGHT OUTER JOIN operation must be a large table.
- Either the left or right table in an INNER JOIN operation can be a large table.
- MAPJOIN cannot be used in a FULL OUTER JOIN operation.
- For MAPJOIN, you can use non-equi joins or combine conditions by using OR. You can leave the ON condition unspecified or use `MAPJOIN ON 1 = 1`, such as `SELECT /* + MAPJOIN(a) */ a.id FROM shop a JOIN table_name b ON 1=1`, to calculate the Cartesian product. However, this calculation method may increase the data amount.
- MaxCompute allows you to specify a maximum of 128 small tables for MAPJOIN. If you specify more than 128 small tables, a syntax error is returned. Separate small tables with commas (,), such as `/*+MAPJOIN(a,b,c)*/`.
- The total memory occupied by small tables cannot exceed 512 MB. MaxCompute compresses your data before storage. As a result, the data amount of small tables sharply increases after they are loaded into the memory. 512 MB indicates the maximum data amount after small tables are loaded into the memory.

Examples

In a MaxCompute SQL statement, you cannot use complex join conditions, such as non-equi joins or the OR logical operator, in the ON condition for a common JOIN operation. However, you can use them for a JOIN operation with MAPJOIN specified.

```
SELECT /* + MAPJOIN(a) */
      a.total_price,
      b.total_price
FROM shop a JOIN sale_detail b
ON a.total_price < b.total_price OR a.total_price + b.total_price < 500;
```

6.5.2.6. SELECT TRANSFORM

6.5.2.6.1. Overview

This topic describes the syntax of the SELECT TRANSFORM statement in MaxCompute.

The SELECT TRANSFORM statement allows you to start a specific child process and insert data in the required format into the child process by using standard input. Then, the SELECT TRANSFORM statement parses the standard output of the child process to obtain the output data. The SELECT TRANSFORM statement allows you to run scripts in other programming languages without the need to write user-defined functions (UDFs).

Syntax:

```
SELECT TRANSFORM(arg1, arg2 ...)
(Row FORMAT DELIMITED (FIELDS TERMINATED BY field_delimiter (ESCAPED BY character_escape))
(LINES SEPARATED BY line_separator)
(NULL DEFINED AS null_value))
USING 'unix_command_line'
(RESOURCES 'res_name' (' 'res_name')*)
( AS col1, col2 ...)
(Row FORMAT DELIMITED (FIELDS TERMINATED BY field_delimiter (ESCAPED BY character_escape))
(LINES SEPARATED BY line_separator) (NULL DEFINED AS null_value))
```

Parameters:

- **SELECT TRANSFORM**: the keyword, which can be replaced with the **MAP** or **REDUCE** keyword. These keywords have the same syntax. We recommend that you use **SELECT TRANSFORM** to make the syntax clearer.
- (arg1, arg2 ...): the input data. The data format is similar to that of the **SELECT** statement. In the default format, the results of expressions for each parameter are implicitly converted into a value of the **STRING** type. Then, the parameters are combined by `\t` and passed to the specified child process.

 **Note** The format is configurable. For more information, see **ROW FORMAT**.

- **USING**: the commands that are used to start a child process. Take note of the following points:
 - In most MaxCompute SQL statements, the **USING** clause specifies resources. However, in the **SELECT TRANSFORM** statement, the **USING** clause specifies the commands to start a child process. The **USING** clause is used to ensure compatibility with the Hive syntax.
 - The syntax of the **USING** clause is similar to that of a shell script. However, instead of running the shell script, the **USING** clause creates a child process based on the input commands. Therefore, some shell features, such as input and output redirection, pipe, and loop, cannot be used. A shell script can be used as the commands to start a child process if necessary.
- **RESOURCES**: the resources that the specified child process can access. You can use one of the following methods to specify resources:
 - Use the **RESOURCES** clause. Example: `using 'sh foo.sh bar.txt' resources 'foo.sh','bar.txt' .`
 - Add the `set odps.sql.session.resources=foo.sh,bar.txt;` flag before SQL statements to specify resources.

 **Note** This is a global configuration. It allows all **SELECT TRANSFORM** statements to access the specified resources. Separate multiple resource files with commas (,).

- **ROW FORMAT**: the input and output formats. The syntax includes two **ROW FORMAT** clauses. The first clause specifies the format of the input data, and the second clause specifies the format of the output data. By default, `\t` is used as a column delimiter, `\n` is used as a row delimiter, and `\N` is used to represent NULL values.

 **Note**

- For `field_delimiter`, `character_escape`, and `line_separator`, only one character is accepted. If you specify a string, the first character in the string takes priority over the others.
- MaxCompute supports syntax in the formats that are specified by Hive, such as `inputRecordReader`, `outputRecordReader`, and `SerDe`. You must enable the Hive-compatible mode to use the syntax. To enable the Hive-compatible mode, add `set odps.sql.hive.compatible=true;` before SQL statements.
- If you specify the syntax that is supported by Hive, such as `inputRecordReader` and `outputRecordReader`, the performance of SQL queries may deteriorate.

- **AS**: the output columns.

Note

- If you do not specify the data types for output columns, the STRING type is used.
- The output data is obtained after the standard output of the child process is parsed. If the data is not of the STRING type, the system implicitly calls the CAST function to convert the data into the STRING type. Exceptions may occur during the conversion process.
- You must specify data types for all output columns.
- If you omit the AS keyword, the field before the first \t in the standard output data is the key and all the following parts are values. This is equivalent to AS(key, value).

6.5.2.6.2. SELECT TRANSFORM examples

6.5.2.6.2.1. Call shell scripts

This topic describes how to use SELECT TRANSFORM to call shell scripts and provides examples.

Assume that you use a shell script to generate 50 rows of data. The values are from 1 to 50. The following code shows the output of the data field:

```
SELECT TRANSFORM(script) USING 'sh' AS (data)
FROM (
    SELECT 'for i in `seq 1 50`; do echo $i; done' AS script
) t
;
```

You can use shell commands as the input for SELECT TRANSFORM.

Note SELECT TRANSFORM allows you to use the scripts of multiple programming languages, such as AWK, Python, Perl, and shell, as the input to implement simple features. You do not need to compile script files or upload resources, which simplifies development.

You can upload a script to implement complex features. For more information, see [Call Python scripts](#).

6.5.2.6.2.2. Call Python scripts

This topic describes how to use SELECT TRANSFORM to call Python scripts and provides examples.

1. Compile a Python script file. In this example, the file name is myplus.py.

```
#!/usr/bin/env python
import sys
line = sys.stdin.readline()
while line:
    token = line.split('\t')
    if (token[0] == '\\N') or (token[1] == '\\N'):
        print '\\N'
    else:
        print str(token[0]) + '\t' + str(token[1])
    line = sys.stdin.readline()
```

2. Add the Python script file as a resource to MaxCompute.

```
add py ./myplus.py -f;
```

3. Execute SELECT TRANSFORM to call the resource.

```

-- Create a test table.
CREATE TABLE testdata(c1 bigint,c2 bigint);
-- Insert test data into the test table.
INSERT INTO TABLE testdata VALUES (1,4),(2,5),(3,6);
-- Execute the following statement:
SELECT
TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py' resources 'myplus.py'
AS (result1,result2)
FROM testdata;
-- The preceding statement is equivalent to the following statement:
set odps.sql.session.resources=myplus.py;
SELECT TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py'
AS (result1,result2)
FROM testdata;

```

Note In MaxCompute, you can use Python scripts as the input for SELECT TRANSFORM. For example, you can run Python commands to call shell scripts.

```
SELECT TRANSFORM('for i in xrange(1, 50): print i;') USING 'python' AS (data);
```

The following result is returned:

```

+-----+-----+
| result1 | result2 |
+-----+-----+
| 1       | 4       |
|        | NULL    |
| 2       | 5       |
|        | NULL    |
| 3       | 6       |
|        | NULL    |
+-----+-----+

```

6.5.2.6.2.3. Call Java scripts

This topic describes how to use SELECT TRANSFORM to call Java scripts and provides examples.

Java scripts are called in a similar way to Python scripts. In this example, you must prepare a Java script file, export it as a JAR package, and then execute the `ADD FILE` statement to add the JAR package as a resource to MaxCompute. Then, the resource can be called by using SELECT TRANSFORM.

1. Prepare a Java script file and export it as a JAR package. In this example, the name of the JAR package is Sum.jar. Sample code:

```

package com.aliyun.odps.test;
import java.util.Scanner
public class Sum {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String s = sc.nextLine();
            String[] tokens = s.split("\t");
            if (tokens.length < 2) {
                throw new RuntimeException("illegal input");
            }
            if (tokens[0].equals("\N") || tokens[1].equals("\N")) {
                System.out.println("\N");
            }
            System.out.println(Long.parseLong(tokens[0]) + Long.parseLong(tokens[1]));
        }
    }
}

```

2. Add the JAR package as a resource to MaxCompute.

```
add jar . /Sum.jar -f;
```

3. Execute the SELECT TRANSFORM statement to call the resource.

```

-- Create a test table.
CREATE TABLE testdata(c1 bigint,c2 bigint);
-- Insert test data into the test table.
INSERT INTO TABLE testdata VALUES (1,4), (2,5), (3,6);
-- Execute the SELECT TRANSFORM statement.
SELECT TRANSFORM(testdata.c1, testdata.c2) USING 'java -cp Sum.jar com.aliyun.odps.test.Sum' resources 'Sum.jar' FROM testdata;
-- The preceding statement is equivalent to the following statement:
set odps.sql.session.resources=Sum.jar;
SELECT TRANSFORM(testdata.c1, testdata.c2) USING 'java -cp Sum.jar com.aliyun.odps.test.Sum' FROM testdata;

```

The following result is returned:

```

+-----+
| cnt |
+-----+
| 5 |
| 7 |
| 9 |
+-----+

```

You can use the preceding method to run most Java utilities.

Although UDTF frameworks are provided for Java and Python, SELECT TRANSFORM makes it easier to compile code. SELECT TRANSFORM is a simpler process because it is not subject to dependencies or format requirements and can be called offline. The paths for Java and Python offline scripts can be obtained from the JAVA_HOME and PYTHON_HOME environment variables.

6.5.2.6.2.4. Call the scripts of other programming languages

This topic describes how to use SELECT TRANSFORM to call the scripts of other programming languages and provides examples.

SELECT TRANSFORM supports the programming languages described in the preceding topics. It also supports commonly used UNIX commands and script interpreters, such as AWK and Perl interpreters.

Example on how to call an AWK script to display the second column:

```
SELECT TRANSFORM(*) USING "awk '{print $2}'" as (data) from testdata;
```

Example on how to call a Perl script:

```
SELECT TRANSFORM (testdata.c1, testdata.c2) USING "perl -e 'WHILE($input = <STDIN>){print $input;}'"
FROM testdata;
```

 **Note** PHP and Ruby are not deployed in MaxCompute compute clusters. Therefore, you cannot call PHP or Ruby scripts in MaxCompute.

6.5.2.6.2.5. Call scripts in series

This topic describes how to use SELECT TRANSFORM to call scripts in series and provides examples.

SELECT TRANSFORM allows you to call scripts in series. For example, you can use DISTRIBUTE BY and SORT BY to pre-process data.

```
SELECT TRANSFORM(key, value) USING 'cmd2' FROM
(
  SELECT TRANSFORM(*) USING 'cmd1' FROM
  (
    SELECT * FROM data DISTRIBUTE BY col2 SORT BY col1
  ) t DISTRIBUTE BY key SORT BY value
) t2;
```

You can also use the MAP or REDUCE keyword to obtain the same results.

```
@a := select * from data distribute by col2 sort by col1;
@b := map * using 'cmd1' distribute by col1 sort by col2 from @a;
reduce * using 'cmd2' from @b;
```

6.5.2.6.3. Performance

This topic describes the advantages of SELECT TRANSFORM and user-defined table-valued functions (UDTFs).

The performance of SELECT TRANSFORM and UDTFs varies based on specific scenarios. In most cases, SELECT TRANSFORM performs better. However, UDTFs perform better as the data amount increases. SELECT TRANSFORM is more suitable for ad hoc data analytics because the development of SELECT TRANSFORM is easier.

The following sections describe the advantages of UDTFs and SELECT TRANSFORM.

Advantages of UDTFs

- For UDTFs, the input parameters and output results support multiple data types. For SELECT TRANSFORM, the child process transfers data based on the standard input and output and processes all data as the STRING type. Therefore, SELECT TRANSFORM requires one more step of data type conversions than UDTFs.
- For SELECT TRANSFORM, data transfer depends on the operating system pipe. The pipe has only a 4 KB cache that cannot be specified. If the operating system pipe is empty or fully occupied, SELECT TRANSFORM cannot respond.
- For UDTFs, the constant parameters do not need to be transferred. However, SELECT TRANSFORM does not

support this feature.

Advantages of SELECT TRANSFORM

- SELECT TRANSFORM supports two processes: parent and child processes. UDTFs support only a single process. If the usage of computing resources is high and the data throughput is low, SELECT TRANSFORM can take advantage of the multi-core feature of the server.
- SELECT TRANSFORM calls underlying systems to read and write data during data transfer. This allows SELECT TRANSFORM to provide better performance in data transfer than Java programs.
- SELECT TRANSFORM supports the native code of tools such as AWK. This allows SELECT TRANSFORM to deliver more performance benefits than Java programs.

6.5.2.7. GROUPING SETS

6.5.2.7.1. Overview

This topic describes GROUPING SETS in MaxCompute.

In some cases, you must execute the UNION ALL clause multiple times to aggregate and analyze data from multiple dimensions. For example, you want to aggregate Column A, aggregate Column B, and then aggregate Column A and Column B. The GROUPING SETS clause is a better choice in these cases.

GROUPING SETS is an extension to the GROUP BY clause in a SELECT statement. GROUPING SETS allows you to group results by using various methods without the need to execute multiple SELECT statements. This allows the MaxCompute engine to generate more efficient execution plans with high performance.

 **Note** Most examples in this topic are demonstrated by using MaxCompute Studio. We recommend that you install MaxCompute Studio before you proceed with subsequent operations. For more information, see [MaxCompute Studio](#).

6.5.2.7.2. Example

This topic provides an example on how to execute the GROUPING SETS statement.

Example:

1. Prepare data.

```
create table requests LIFECYCLE 20 as
select * from values
  (1, 'windows', 'PC', 'Beijing'),
  (2, 'windows', 'PC', 'Shijiazhuang'),
  (3, 'linux', 'Phone', 'Beijing'),
  (4, 'windows', 'PC', 'Beijing'),
  (5, 'ios', 'Phone', 'Shijiazhuang'),
  (6, 'linux', 'PC', 'Beijing'),
  (7, 'windows', 'Phone', 'Shijiazhuang')
as t(id, os, device, city);
```

2. Execute the GROUPING SETS statement.

```
SELECT os,device, city ,COUNT(*)
FROM requests
GROUP BY os, device, city GROUPING SETS((os, device), (city), ());
```

The following result is returned:

```

+----+-----+-----+-----+
| os | device | city | cnt      |
+----+-----+-----+-----+
| NULL | NULL   | NULL | 7        |
| NULL | NULL   | Beijing | 4      |
| NULL | NULL   | Shijiazhuang | 3    |
| ios | Phone  | NULL | 1        |
| linux | PC     | NULL | 1        |
| linux | Phone  | NULL | 1        |
| windows | PC    | NULL | 3        |
| windows | Phone | NULL | 1        |
+----+-----+-----+-----+

```

You can also execute multiple `SELECT` statements to obtain the same result.

```

SELECT NULL, NULL, NULL, COUNT(*)
FROM requests
UNION ALL
SELECT os, device, NULL, COUNT(*)
FROM requests GROUP BY os, device
UNION ALL
SELECT null, null, city, COUNT(*)
FROM requests GROUP BY city;

```

However, the execution of `GROUPING SETS` is simpler and more efficient.

Note NULL is used as placeholders for the expressions that are not used in `GROUPING SETS`. This way, you can perform `UNION` operations on the result sets.

6.5.2.7.3. CUBE and ROLLUP

This topic describes the `CUBE` and `ROLLUP` functions for `GROUPING SETS`.

`CUBE` and `ROLLUP` are special `GROUPING SETS` functions. `CUBE` lists all the possible combinations of specific columns as grouping sets. `ROLLUP` aggregates data by level to generate grouping sets.

Examples:

```

GROUP BY CUBE(a, b, c)
-- Equivalent to the following statement:
GROUPING SETS((a,b,c), (a,b), (a,c), (b,c), (a), (b), (c), ())
GROUP BY ROLLUP(a, b, c)
-- Equivalent to the following statement:
GROUPING SETS((a,b,c), (a,b), (a), ())
GROUP BY CUBE ( (a, b), (c, d) )
-- Equivalent to the following statement:
GROUPING SETS (
    ( a, b, c, d ),
    ( a, b      ),
    (      c, d ),
    (          )
)
GROUP BY ROLLUP ( a, (b, c), d )
-- Equivalent to the following statement:
GROUPING SETS (
    ( a, b, c, d ),
    ( a, b, c   ),
    ( a        ),
    (          )
)
GROUP BY a, CUBE (b, c), GROUPING SETS ((d), (e))
-- Equivalent to the following statement:
GROUP BY GROUPING SETS (
    (a, b, c, d), (a, b, c, e),
    (a, b, d),   (a, b, e),
    (a, c, d),   (a, c, e),
    (a, d),     (a, e)
)
GROUP BY grouping sets((b), (c),rollup(a,b,c))
-- Equivalent to the following statement:
GROUP BY GROUPING SETS (
    (b), (c),
    (a,b,c), (a,b), (a), ()
)

```

6.5.2.7.4. GROUPING and GROUPING_ID

This topic describes the GROUPING and GROUPING_ID functions in GROUPING SETS.

NULL is used as placeholders in the results of GROUPING SETS. As a result, the NULL placeholders cannot be distinguished from the NULL values. To address this issue, MaxCompute provides the GROUPING function.

GROUPING allows you to specify the name of a column as a parameter. If specific rows are aggregated based on the column specified in the GROUPING function, 0 is returned, which indicates that NULL is an input value. Otherwise, 1 is returned, which indicates that NULL is a placeholder in GROUPING SETS.

GROUPING_ID allows you to specify the names of one or more columns as parameters. The GROUPING results of these columns are formed into integers by using bit map.

Examples:

```

SELECT a,b,c ,COUNT(*),
GROUPING(a) ga, GROUPING(b) gb, GROUPING(c) gc, GROUPING_ID(a,b,c) groupingid
FROM VALUES (1,2,3) as t(a,b,c)
GROUP BY CUBE(a,b,c);

```

The following result is returned:

```

+-----+-----+-----+-----+-----+-----+-----+-----+
----+
| a      | b      | c      | _c3    | ga     | gb     | gc     | groupin
gid |
+-----+-----+-----+-----+-----+-----+-----+-----+
----+
| NULL   | NULL   | NULL   | 1      | 1      | 1      | 1      | 7
|
| NULL   | NULL   | 3      | 1      | 1      | 1      | 0      | 6
|
| NULL   | 2      | NULL   | 1      | 1      | 0      | 1      | 5
|
| NULL   | 2      | 3      | 1      | 1      | 0      | 0      | 4
|
| 1      | NULL   | NULL   | 1      | 0      | 1      | 1      | 3
|
| 1      | NULL   | 3      | 1      | 0      | 1      | 0      | 2
|
| 1      | 2      | NULL   | 1      | 0      | 0      | 1      | 1
|
| 1      | 2      | 3      | 1      | 0      | 0      | 0      | 0
|
+-----+-----+-----+-----+-----+-----+-----+-----+
----+

```

By default, the columns that are not used in GROUP BY are filled with NULL. You can use the GROUPING function to obtain more useful data.

```

SELECT
  IF(GROUPING(os) == 0, os, 'ALL') as os,
  IF(GROUPING(device) == 0, device, 'ALL') as device,
  IF(GROUPING(city) == 0, city, 'ALL') as city ,
  COUNT(*) as count
FROM requests
GROUP BY os, device, city GROUPING SETS((os, device), (city), ());

```

The following figure shows the returned result.

	os	device	city	count
1	ALL	ALL	ALL	7
2	ALL	ALL	Beijing	4
3	ALL	ALL	Shijiazhuang	3
4	ios	Phone	ALL	1
5	linux	PC	ALL	1
6	linux	Phone	ALL	1
7	windows	PC	ALL	3
8	windows	Phone	ALL	1

6.5.2.8. UNION, INTERSECT, and EXCEPT

This topic describes the SQL statements that use the UNION, INTERSECT, and EXCEPT set operators, such as UNION, UNION ALL, UNION DISTINCT, INTERSECT, INTERSECT ALL, INTERSECT DISTINCT, EXCEPT, EXCEPT ALL, and EXCEPT DISTINCT.

Syntax:

```

select_statement UNION ALL select_statement;
select_statement UNION [DISTINCT] select_statement;
select_statement INTERSECT ALL select_statement;
select_statement INTERSECT [DISTINCT] select_statement;
select_statement EXCEPT ALL select_statement;
select_statement EXCEPT [DISTINCT] select_statement;
select_statement MINUS ALL select_statement;
select_statement MINUS [DISTINCT] select_statement;

```

Parameters:

- **UNION**: the union of two datasets. The UNION operation combines two datasets into one dataset.
- **INTERSECT**: the intersection of two datasets. The INTERSECT operation returns the records contained in both datasets.
- **EXCEPT**: the complement of the second dataset in the first dataset. The EXCEPT operation returns the records that are contained in the first dataset, but not in the second dataset.
- **MINUS**: equivalent to EXCEPT.

Examples:

UNION:

- If UNION ALL is used, all records of the two datasets are returned.

```

SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)
UNION ALL
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);

```

The following result is returned:

```

+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
| 1      | 4      |
| 1      | 2      |
| 1      | 2      |
| 3      | 4      |
+-----+-----+

```

- If multiple UNION ALL clauses are used, parentheses can be used to prioritize the clauses.

```

SELECT * FROM src UNION ALL (SELECT * FROM src2 UNION ALL SELECT * FROM src3);

```

- If UNION is used, duplicate records are not included in the returned records. UNION is equivalent to UNION DISTINCT.

```

SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)
UNION
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);
-- The preceding statement is equivalent to the following statement:
SELECT DISTINCT * FROM (<Result of UNION ALL>)t;

```

The following result is returned:

```

+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
| 1      | 4      |
| 3      | 4      |
+-----+-----+

```

- Assume that UNION is followed by a CLUSTER BY, DISTRIBUTED BY, SORT BY, ORDER BY, or LIMIT clause. If `set odps.sql.type.system.odps2=false;` is used, the clause works on the result of the last select_statement after the UNION operator. If `odps.sql.type.system.odps2=true;` is used, the clause works on the result of all the UNION operations.

```

set odps.sql.type.system.odps2=true;
SELECT explode(array(3, 1)) AS (a) UNION ALL SELECT explode(array(0, 4, 2)) AS (a) ORDER BY a LIMIT 3;

```

The following result is returned:

```

+-----+
| a      |
+-----+
| 0      |
| 1      |
| 2      |
+-----+

```

INTERSECT:

- INTERSECT ALL:

```

SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
INTERSECT ALL
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);

```

The following result is returned:

```

+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
| 1      | 2      |
| 3      | 4      |
+-----+-----+

```

- INTERSECT DISTINCT:

```

SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
INTERSECT DISTINCT
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);

```

The following result is returned. The preceding statement is equivalent to the `SELECT DISTINCT * FROM (< Result of INTERSECT ALL >) t;` statement.

```

+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
| 3      | 4      |
+-----+-----+
    
```

EXCEPT:

- EXCEPT ALL:

```

SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
EXCEPT ALL
SELECT * FROM VALUES (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
    
```

The following result is returned:

```

+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
| 1      | 2      |
| 3      | 4      |
| 7      | 8      |
+-----+-----+
    
```

- EXCEPT DISTINCT:

```

SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
EXCEPT
SELECT * FROM VALUES (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
    
```

The following result is returned. The preceding statement is equivalent to the `SELECT DISTINCT * FROM left_branch EXCEPT ALL SELECT DISTINCT * FROM right_branch;` statement.

```

+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
| 7      | 8      |
+-----+-----+
    
```

Note

- Set operations do not necessarily return sorted results.
- The two branches for a set operation must have the same number of output columns with consistent data types. If data types are inconsistent, implicit conversion of the data types may be performed. To ensure compatibility for set operations, MaxCompute forbids the implicit conversion between STRING and other data types.
- MaxCompute supports a maximum of 256 branches for a set operation. If the limit is exceeded, an error is returned.

6.5.3. EXPLAIN

This topic describes the EXPLAIN statement in MaxCompute SQL.

This statement shows the structure of the execution plan of a DML statement. An execution plan is a program that executes SQL statements.

Syntax:

```
EXPLAIN <DML query>;
```

Note

The execution results of an EXPLAIN statement include:

- Dependencies among all the jobs of this DML statement
- Dependencies among all the tasks of each job
- Dependencies among all operators in a task

Examples:

```
EXPLAIN
SELECT abs(a.key), b.value FROM src a JOIN src1 b ON a.value = b.value;
```

The output of the EXPLAIN statement consists of the following three parts:

- **Job dependencies:** `job0 is root job`. This query requires only one job (job0). Therefore, only one row of information is displayed.
- **Task dependencies.** Example:

```
In Job job0:
root Tasks: M1_Stg1, M2_Stg1
J3_1_2_Stg1 depends on: M1_Stg1, M2_Stg1
```

Note

- Job0 contains three tasks, M1_Stg1, M2_Stg1, and J3_1_2_Stg1. J3_1_2_Stg1 is executed after the M1_Stg1 and M2_Stg1 tasks are complete.
- Naming conventions for a task: MaxCompute provides the task types of map, reduce, join, and local work. The first letter in a task name indicates the type of the task. For example, M2Stg1 is a map task. The number following the first letter indicates the task ID. This ID is unique among all tasks that correspond to the current query. Digits that are combined by underscores (_) indicate the task dependencies. For example, J3_1_2_Stg1 indicates that the current task with the ID of 3 depends on two tasks whose IDs are 1 and 2.

- Operator structure, where each operator string describes the execution semantics of a task.

```

In Task M1_Stgl:
  Data source: yudi_2.src          # Data source describes the input of the task.
  TS: alias: a                    # TableScanOperator
    RS: order: +                  # ReduceSinkOperator
      keys:
        a.value
      values:
        a.key
      partitions:
        a.value
In Task J3_1_2_Stgl:
  JOIN: a INNER JOIN b           # JoinOperator
      SEL: Abs(UDFToDouble(a._col0)), b._col5 # SelectOperator
      FS: output: None           # FileSinkOperator
In Task M2_Stgl:
  Data source: yudi_2.src1
  TS: alias: b
    RS: order: +
      keys:
        b.value
      values:
        b.value
      partitions:
        b.value

```

The following table describes the operators.

Operator	Description
TableScanOperator	Describes the logic of FROM statement blocks in a query statement. The alias of the input table is displayed in the EXPLAIN results.
SelectOperator	Describes the logic of SELECT statement blocks in a query statement. The columns that are transferred to the next operator are displayed in the EXPLAIN results. Separate multiple columns with commas (,). <ul style="list-style-type: none"> ◦ If columns are referenced, the value is in the < alias >.< column_name > format. ◦ If the result of an expression is transferred, the value is displayed as a function, such as func1(arg1_1, arg1_2, func2(arg2_1, arg2_2)). ◦ If constants are transferred, the value is immediately displayed.
FilterOperator	Describes the logic of WHERE statement blocks in a query statement. The EXPLAIN results include a WHERE clause, with the display rules that are similar to those of SelectOperator.
JoinOperator	Describes the logic of JOIN statement blocks in a query statement. The EXPLAIN results show which tables are joined in which way.
GroupByOperator	Describes the logic of aggregate operations. This operator is displayed if an aggregate function is used in a query. The content of the aggregate function is displayed in the EXPLAIN results.
ReduceSinkOperator	Describes the logic of data distribution between tasks. If the result of the current task is transferred to another task, ReduceSinkOperator must be used to distribute data at the last stage of the current task. The sorting method of output data records, the distributed keys and values, and the columns used to calculate the hash value are displayed in the EXPLAIN results.

Operator	Description
FileSinkOperator	Describes the logic of storage operations on final data records. If an INSERT statement block exists in a query, the name of the required table is displayed in the EXPLAIN results.
LimitOperator	Describes the logic of LIMIT statement blocks in a query statement. The number of records specified by LIMIT is displayed in the EXPLAIN results.
MapjoinOperator	Describes JOIN operations on large tables. This operator is similar to JoinOperator.

 **Note**

- If a query statement is complex, the EXPLAIN statement returns excessive results. As a result, API limits are reached and complete results cannot be obtained. In this case, you can split a query and execute the EXPLAIN statement on each subquery to understand the structure of the job.
- If you specify more than 10,000 partitions in a query, large amounts of source data needs to be read. To circumvent this limit, you can add filter conditions in the query to filter out most partitions.

6.5.4. IF statement

This topic describes the IF statement that MaxCompute SQL supports.

MaxCompute SQL supports the IF-ELSE statement. You can use the IF-ELSE statement to execute SQL scripts with specific conditions. A condition in the IF-ELSE statement can be a standard variable or a scalar subquery that returns only one column value from one row.

The IF statement allows the system to automatically select the execution logic based on the specified conditions. MaxCompute supports the following IF syntax:

```
IF (condition) BEGIN
  statement 1
  statement 2
  ...
END
IF (condition) BEGIN
  statements
END ELSE IF (condition2) BEGIN
  statements
END ELSE BEGIN
  statements
END
```

 **Note** The BEGIN and END conditional clause can be omitted if it contains only one statement, similar to '{ }' in Java.

The IF statement can contain two types of conditions: expressions and scalar subqueries. Both of them are of the BOOLEAN type.

- Expressions: A BOOLEAN-type expression in the IF-ELSE statement determines which branch is executed at the compiling stage. Example:

```

@date := '20190101';
@row TABLE(id STRING); -- Declare the row variable. The type of the row is Table and schema is STRING.
IF ( cast(@date as bigint) % 2 == 0 ) BEGIN
@row := SELECT id from src1;
END ELSE BEGIN
@row := SELECT id from src2;
END
INSERT OVERWRITE TABLE dest SELECT * FROM @row;

```

- Scalar subqueries: A BOOLEAN-type scalar subquery in the IF-ELSE statement determines which branch is executed at the running stage. Therefore, you must submit multiple jobs. Example:

```

@i bigint;
@t table(id bigint, value bigint);
IF ((SELECT count(*) FROM src WHERE a = '5') > 1) BEGIN
@i := 1;
@t := select @i, @i*2;
END ELSE
BEGIN
@i := 2;
@t := select @i, @i*2;
END
select id, value from @t;

```

6.5.5. UPDATE and DELETE

This topic describes the UPDATE and DELETE statements in MaxCompute SQL.

The UPDATE and DELETE statements are used to manage data of specific rows in tables or partitions.

Background information

MaxCompute data manipulation language (DML) statements have limits on table operations. MaxCompute allows you to use only the INSERT OVERWRITE and INSERT INTO statements to delete and update some rows in a table. For example, if you use an INSERT statement to modify small amounts of data in a table or partition, you must use a SELECT statement to read all data and modify the data. Then, you can execute the INSERT statement to insert the modified data into the table or partition. This is a low-efficiency process. However, if you use the UPDATE and DELETE statements, the amount of data read or written in a query significantly decreases.

History tables were considered the best to perform multiple UPDATE operations at a time. If you use a history table, you must add auxiliary columns, such as start_date and end_date, in the table. These columns indicate the lifecycle of the data records in a row. To query the current status of a table, you must find the latest status from large amounts of data based on the timestamp. This is a time-consuming process. However, the UPDATE and DELETE statements directly read data from the current table.

Description

When you execute a DELETE statement, the system automatically generates a Delta file. txnid(bigint) and rowid(bigint) in the Delta file indicate which rows in which transactional table are deleted from the Base file.

 **Note** The Delta file stores the row numbers that are used to indicate the deleted rows in the Base file.

If you execute a DELETE statement again, another Delta file named f2.del is generated. The f2.del file also stores the row numbers based on the Base file. During data reading, all Delta files are queried and data that is not deleted is returned.

Similarly, the UPDATE statement is converted into the combination of the DELETE and INSERT INTO statements.

If jobs are run in parallel and the tables on which you want to perform operations are the same, conflicts may occur. For more information, see [ACID semantics of MaxCompute concurrent write jobs](#).

Precautions

- The UPDATE and DELETE operations must be performed based on transactional tables. For more information about how to create a transactional table, see [Create a table](#).
- When you execute an UPDATE statement, you must add `set odps.service.mode=off;` before the statement and commit them at the same time.
- If the ratio of the rows that you want to delete or update to all rows is small, the performance of the data that is read from a table is slightly affected. In this case, we recommend that you use the UPDATE or DELETE statement. If the ratio is less than 5%, we recommend that you use the UPDATE or DELETE statement. The maximum ratio varies based on your business scenarios.
- If you perform the UPDATE or DELETE operation multiple times to update or delete a small number of rows in a table, we recommend that you execute the COMPACT statement after the UPDATE and DELETE statements. This reduces the storage usage of the table.
- If you perform less UPDATE or DELETE operations to update or delete a large number of rows in a table and will frequently read data from the table, we recommend that you use the INSERT OVERWRITE and INSERT INTO statements. For example, a total of 10% of the data in a table needs to be updated or deleted ten times a day. In this case, we recommend that you evaluate the sum of the costs of the UPDATE or DELETE statement and the read performance consumption based on the generated Delta file. Then, compare it with that of the INSERT OVERWRITE and INSERT INTO statements to select an efficient method.
- After you delete data, a Delta file is generated. As a result, the storage resources may not be released. If you want to use the DELETE statement to delete data and release the storage resources, you can use the ALTER TABLE COMPACT statement to merge the Base files with Delta files.

Limits

- If you use the ALTER TABLE COMPACT statement in MaxCompute SQL mode of DataWorks, a compatibility issue occurs. To handle this issue, we recommend that you use the MaxCompute client whose version must be **0.35.0**.
- You can specify the transactional attribute for a table only when you create the table. However, you are not allowed to modify the transactional attribute of an existing table by using the ALTER TABLE COMPACT statement. If you modify the transactional attribute of an existing table, an error is returned.

```
alter table not_txn_tbl set tblproperties("transactional"="true");
-- FAILED is returned. Catalog Service Failed, Error Code: 151, Error Message: Set transactional i
s not supported.
```

- You cannot specify clustered tables or external tables as transactional tables.
- Existing internal tables or external tables cannot be converted into transactional tables, and transactional tables cannot be converted into standard tables.
- Tasks from peripheral systems, such as Graph and Machine Learning Platform for AI (PAI), are not supported. If these tasks access a transactional table, an error is returned.
- CLONE TABLE, MERGE PARTITION, and incremental replication operations cannot be performed on transactional tables.
- Before you perform the UPDATE, DELETE, or INSERT OVERWRITE operation on data in transactional tables, make sure that you back up the data in advance.
- You must have the Select and Update permissions on a table to perform the UPDATE and DELETE operations.

DELETE

Syntax:

```
delete from <table_name> [WHERE where_condition];
```

Description: This statement is used to delete some rows from a table.

The following examples are provided:

Example 1: Delete some rows from a table.

```
-- Create a transactional table.
create table if not exists acid_delete(id bigint) tblproperties ("transactional"="true");
-- Insert data into the table.
insert overwrite table acid_delete values(1),(2),(3);
-- Query the table to check whether data is inserted.
select * from acid_delete;
```

The following result is returned:

```
+-----+
| id    |
+-----+
| 1     |
| 2     |
| 3     |
+-----+
```

```
-- Delete the row whose id value is 2. You must enter Yes or No to confirm the operation in the MaxC
ompute client.
delete from acid_delete where id = 2;
-- Read data from the table. Only the rows whose id values are 1 and 3 are retained in the table.
select * from acid_delete;
```

The following result is returned:

```
+-----+
| id    |
+-----+
| 1     |
| 3     |
+-----+
```

Example 2: Delete some rows from a partitioned table.

```
-- Create a transactional table.
create table if not exists acid_delete_pt(id bigint) partitioned by(ds string) tblproperties ("trans
actional"="true");
-- Add partitions to the table.
alter table acid_delete_pt add if not exists partition (ds= '2019');
-- Insert data into the partitions.
insert overwrite table acid_delete_pt partition (ds='2019') values(1),(2),(3);
-- Query the table to check whether data is inserted.
select * from acid_delete_pt where ds = '2019';
```

The following result is returned:

```
+-----+-----+
| id   | ds   |
+-----+-----+
| 1    | 2019 |
| 2    | 2019 |
| 3    | 2019 |
+-----+-----+
```

```
-- Delete data from the partitions. You must enter Yes or No to confirm the operation in the MaxCompute client.
delete from acid_delete_pt where ds='2019' and id= 2;
-- Read data from the table. Only the rows whose id values are 1 and 3 are retained in the table.
select * from acid_delete_pt where ds = '2019';
```

The following result is returned:

```
+-----+-----+
| id   | ds   |
+-----+-----+
| 1    | 2019 |
| 3    | 2019 |
+-----+-----+
```

UPDATE

Syntax:

```
update <table_name> set col1 = value1 [, col2 = value2 ...] [WHERE where_condition];
update <table_name> set (col1 [, col2 ...]) = (value1 [, value2 ...]);
```

Description: This statement is used to update some rows in a table.

 **Note** The UPDATE statement is converted into the combination of the DELETE and INSERT INTO statements to update some rows.

The following examples are provided:

Example 1: Update some rows in a table.

```
-- Create a transactional table.
create table if not exists acid_update(id bigint) tblproperties ("transactional"="true");
-- Insert data into the table.
insert overwrite table acid_update values(1),(2),(3);
-- Query the table to check whether data is inserted.
select * from acid_update;
```

The following result is returned:

```
+-----+
| id   |
+-----+
| 1    |
| 2    |
| 3    |
+-----+
```

```
-- Update the value 2 in the id column to 4.
set odps.service.mode=off;
update acid_update set id = 4 where id = 2;
-- Query the table to check whether the value 2 is updated to 4 in the id column.
select * from acid_update;
```

The following result is returned:

```
+-----+
| id    |
+-----+
| 4     |
| 1     |
| 3     |
+-----+
```

Example 2: Update some rows in a partitioned table.

```
-- Create a transactional table.
create table if not exists acid_update_pt(id bigint) partitioned by(ds string) tblproperties ("trans
actional"="true");-- Add a partition.
-- Add partitions to the table.
alter table acid_update_pt add if not exists partition (ds= '2019');
-- Insert data into the partitions.
insert overwrite table acid_update_pt partition (ds='2019') values(1),(2),(3);
-- Query the table to check whether data is inserted.
select * from acid_update_pt where ds = '2019';
```

The following result is returned:

```
+-----+-----+
| id   | ds   |
+-----+-----+
| 1    | 2019 |
| 2    | 2019 |
| 3    | 2019 |
+-----+-----+
```

```
-- Update the id values of rows in partition 2019 to 4. The rows must have the id values of 2.
set odps.service.mode=off;
update acid_update_pt set id = 4 where ds = '2019' and id = 2;
-- Query the table to check whether the value 2 is updated to 4 in the id column.
select * from acid_update_pt where ds = '2019';
```

The following result is returned:

```
+-----+-----+
| id   | ds   |
+-----+-----+
| 4    | 2019 |
| 1    | 2019 |
| 3    | 2019 |
+-----+-----+
```

Example 3: Update several columns and update a destination table by using the data in a source table.

```

-- Create a destination table named acid_update_t and create a table named acid_update_s for joins.
create table if not exists acid_update_t(id bigint,value1 bigint,value2 bigint) tblproperties ("transactional"="true");
create table if not exists acid_update_s(id bigint,value1 bigint,value2 bigint);
-- Insert data into these tables.
insert overwrite table acid_update_t values(2,20,21),(3,30,31),(4,40,41);
insert overwrite table acid_update_s values(1,100,101),(2,200,201),(3,300,301);
-- Use constants to update the acid_update_t table.
set odps.service.mode=off;
update acid_update_t set (value1, value2) = (3, 4);
-- Use the data in the acid_update_s table to update the acid_update_t table. Left join the acid_update_t table with the acid_update_s table. If you want to update only the intersection of the two tables, you must add a WHERE clause to the update <table_name> statement.
set odps.service.mode=off;
update acid_update_t set (value1, value2) = (select value1, value2 from acid_update_s where acid_update_t.id = acid_update_s.id);
-- Use the data in the acid_update_s table to update the acid_update_t table. Add filter conditions for the acid_update_t table to update only the intersection of the two tables.
set odps.service.mode=off;
update acid_update_t set (value1, value2) = (select value1, value2 from acid_update_s where acid_update_t.id = acid_update_s.id) where acid_update_t.id in (select id from acid_update_s);
-- Use the aggregate results of the acid_update_t and acid_update_s tables to update the acid_update_t table.
set odps.service.mode=off;
update acid_update_t set (id, value1, value2) = (select id, max(value1),max(value2) from acid_update_s where acid_update_t.id = acid_update_s.id group by acid_update_s.id) where acid_update_t.id in (select id from acid_update_s);

```

MERGE INTO

The MERGE INTO statement encapsulates the INSERT, UPDATE, and DELETE operations. You can use this statement to perform multiple operations on a destination table after the table is joined with a source table.

MaxCompute DML provides the UPDATE and DELETE statements to enrich its expressions. However, if you want to use several INSERT, UPDATE, and DELETE statements to perform multiple UPDATE operations on a destination table at a time, you must execute several statements. These statements trigger multiple full table scans. The MERGE INTO statement requires only one full table scan for you to complete the operations. The MERGE INTO statement is more efficient than the INSERT, UPDATE, and DELETE statements.

If you use the INSERT, UPDATE, and DELETE statements in a job to update a table, the update succeeds only when all the statements are successfully executed. If only some of these statements are successfully executed, the execution cannot be rolled back. However, the MERGE INTO statement is atomic, which can be used to prevent this issue.

Syntax:

```

MERGE INTO <target table> AS T USING <source expression/table> AS S
ON <boolean expression1>
WHEN MATCHED [AND <boolean expression2>] THEN UPDATE SET <set clause list>
WHEN MATCHED [AND <boolean expression3>] THEN DELETE
WHEN NOT MATCHED [AND <boolean expression4>] THEN INSERT VALUES <value list>

```

Description: This statement is used to update or insert data from the source expression or table into the destination table. This statement is also used to delete data from the source expression or table from the destination table.

Parameters:

- **source expression/table:** the source table. The value of this parameter can be a table, view, or subquery.

- target table: the destination table, which must be an existing physical table.
- ON: the clause that is used to determine whether the source table is joined with the destination table. boolean expression specifies the join condition.
- WHEN MATCHED...THEN: specifies the action when the ON clause determines that the two tables are joined. Data obtained from multiple WHEN MATCHED [AND <boolean expression>] clauses must have no intersections. The MERGE INTO statement has clear objectives. You are not allowed to perform the INSERT or UPDATE operation on the same row in a single MERGE INTO statement. When you execute the MERGE INTO statement, make sure that the data in the source tables that meet the ON joint condition is unique. Otherwise, an error is returned.

 Note

- If a MERGE INTO statement has three WHEN clauses, the UPDATE, DELETE, and INSERT operations can appear at most once in each clause.
- WHEN NOT MATCHED must be the last WHEN clause and can be used only for an INSERT operation.
- If both the UPDATE and DELETE statements are included in a MERGE INTO statement, the statement that appears first must include [AND <boolean expression>].

The following example is provided:

Prepare the test data.

```
-- Create a destination table.
create table if not exists acid_address_book_base1
(id bigint,first_name string,last_name string,phone string)
partitioned by(year string, month string, day string, hour string)
tblproperties ("transactional"="true");
-- Create a source table.
create table if not exists tmp_table1
(id bigint, first_name string, last_name string, phone string, _event_type_string);
-- Insert the test data into the destination table.
insert overwrite table acid_address_book_base1
partition(year='2020', month='08', day='20', hour='16')
values (4, 'nihaho', 'li', '222'), (5, 'tahao', 'ha', '333'),
(7, 'djh', 'hahh', '555');
-- Insert the test data into the source table.
insert overwrite table tmp_table1 values
(1, 'hh', 'liu', '999', 'I'), (2, 'cc', 'zhang', '888', 'I'),
(3, 'cy', 'zhang', '666', 'I'), (4, 'hh', 'liu', '999', 'U'),
(5, 'cc', 'zhang', '888', 'U'), (6, 'cy', 'zhang', '666', 'U');
```

If data meets the ON joint condition and is from the source table, you can use the data to perform the MERGE INTO operation on the destination table. If data does not meet the ON joint condition but is of the I type specified by event_type in the source table, you can insert the data into the destination table.

```
merge into acid_address_book_base1 as t using tmp_table1 as s
on s.id = t.id and t.year='2020' and t.month='08' and t.day='20' and t.hour='16'
when matched
then update set t.first_name = s.first_name, t.last_name = s.last_name, t.phone = s.phone
when not matched and (s._event_type_='I')
then insert values(s.id, s.first_name, s.last_name,s.phone,'2020','08','20','16');
-- Query the destination table after the preceding operations are complete.
select * from acid_address_book_base1;
```

The following result is returned:

id	first_name	last_name	phone	year	month	day	hour
4	hh	liu	999	2020	08	20	16
5	cc	zhang	888	2020	08	20	16
7	djh	hahh	555	2020	08	20	16
1	hh	liu	999	2020	08	20	16
2	cc	zhang	888	2020	08	20	16
3	cy	zhang	666	2020	08	20	16

ALTER TABLE COMPACT

If you perform DELETE operations on a transactional table, the Base file that you want to delete is not modified. A Delta file is generated every time you perform a DELETE operation. As a result, if you perform multiple DELETE operations, the generated Delta files occupy more storage resources.

If you perform several DELETE operations on the same table or partition, a large number of Delta files may be generated based on the amount of the data that you delete. Similarly, if you perform an UPDATE operation which is converted into the combination of the DELETE and INSERT INTO operations, Delta files are also generated. If the system reads data from the table, it must load the generated Delta files to find the deleted rows. A large number of Delta files reduce data reading efficiency.

In this case, you can use the ALTER TABLE COMPACT statement to merge the Base files with Delta files to reduce storage usage and improve data reading efficiency.

Syntax:

```
alter table <table_name> [partition (partition_key = 'partition_value' [, ...])] compact [major|minor];
```

Description: This statement is used to merge the Base files with Delta files to reduce storage usage and improve data reading efficiency.

- Minor compaction: merges all Delta files with their Base files. The Delta files must be generated based on the same Base file. Delta files that are generated based on different Base files cannot be merged.
- Major compaction: merges all Delta files generated based on the same Base file with the Base file, deletes the Delta files, and merges small files in multiple Base files that correspond to tables. If the size of the Base file is less than 32 MB or a Delta file is generated, the INSERT OVERWRITE operation is performed on the table. However, if the size of the Base file is greater than or equal to 32 MB or no Delta files are generated, the INSERT OVERWRITE operation is not performed on the table.

Sample statements:

```
alter table acid_delete compact minor;
alter table acid_update_pt partition (ds = '2019') compact major;
```

The following result is returned:

```
Summary:
table name: acid_update_pt /ds=2019 instance count: 2 run time: 16
before merge, file count: 8 file size: 2423 file physical size: 7269
after merge, file count: 2 file size: 642 file physical size: 1926
```

6.6. Built-in functions

6.6.1. Mathematical functions

6.6.1.1. ABS

This topic describes the ABS function in mathematical functions and provides examples.

Syntax:

```
double abs(double number)
bigint abs(bigint number)
decimal abs(decimal number)
```

Description: This function calculates the absolute value of number.

Parameters: The number parameter supports a value of the BIGINT, DOUBLE, or DECIMAL type. If the input value is of the BIGINT, DOUBLE, or DECIMAL type, a value of the same type is returned. If the input value is of the STRING type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: The type of the return value depends on that of the input value, which can be DOUBLE, BIGINT, or DECIMAL. If the input value is NULL, NULL is returned.

 **Note** If the input value is of the BIGINT type and is greater than the maximum value of the BIGINT type, a value of the DOUBLE type is returned. However, the precision may be lost.

Examples:

```
abs(null)=null
abs(-1)=1
abs(-1.2)=1.2
abs("-2")=2.0
abs(122320837456298376592387456923748)=1.2232083745629837e32
```

The following example shows the usage of an ABS function in SQL statements. Other built-in functions, except window functions and aggregate functions, are used in a similar way.

```
select abs(id) from tbl1;
-- Calculate the absolute value of the id field in tbl1.
```

6.6.1.2. ACOS

This topic describes the ACOS function in mathematical functions and provides examples.

Syntax:

```
double acos(double number)
decimal acos(decimal number)
```

Description: This function calculates the arccosine of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. The value ranges from -1 to 1. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. The value ranges from 0 to π . If the input value is NULL, NULL is returned.

Examples:

```
acos("0.87") = 0.5155940062460905
acos(0) = 1.5707963267948966
```

6.6.1.3. ASIN

This topic describes the ASIN function in mathematical functions and provides examples.

Syntax:

```
double asin(double number)
decimal asin(decimal number)
```

Description: This function calculates the arcsine of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. The value ranges from -1 to 1. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. The value ranges from $-\pi/2$ to $\pi/2$. If the input value is NULL, NULL is returned.

Examples:

```
asin(1) = 1.5707963267948966
asin(-1) = -1.5707963267948966
```

6.6.1.4. ATAN

This topic describes the ATAN function in mathematical functions and provides examples.

Syntax:

```
double atan(double number)
```

Description: This function calculates the arctangent of number.

Parameters: The number parameter supports a value of the DOUBLE type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE type is returned. The value ranges from $-\pi/2$ to $\pi/2$. If the input value is NULL, NULL is returned.

Examples:

```
atan(1) = 0.7853981633974483;
atan(-1) = -0.7853981633974483
```

6.6.1.5. CEIL

This topic describes the CEIL function in mathematical functions and provides examples.

Syntax:

```
bigint ceil(double value)
bigint ceil(decimal value)
```

Description: This function rounds up value and returns the nearest integer.

Parameters: The value parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the BIGINT type is returned. If the input value is NULL, NULL is returned.

Examples:

```
ceil(1.1) = 2
ceil(-1.1) = -1
```

6.6.1.6. CONV

This topic describes the CONV function in mathematical functions and provides examples.

Syntax:

```
string conv(string input, bigint from_base, bigint to_base)
```

Description: This function converts a number from one number system to another.

Parameters:

- input: the integer you want to convert, which is of the STRING type. If the input value is of the BIGINT or DOUBLE type, it is implicitly converted to a value of the STRING type before calculation.
- from_base, to_base: decimal numbers. The values can be 2, 8, 10, or 16. If the input value is of the STRING or DOUBLE type, it is implicitly converted to a value of the BIGINT type before calculation.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned. The conversion process runs at 64-bit precision. If an overflow occurs, an error is returned. If the input value is a negative value that begins with an en dash (-), an error is returned. If the input value is a decimal, it is converted to an integer before the conversion of number systems. The decimal part is left out.

Examples:

```
conv('1100', 2, 10) = '12'
conv('1100', 2, 16) = 'c'
conv('ab', 16, 10) = '171'
conv('ab', 16, 16) = 'ab'
```

6.6.1.7. COS

This topic describes the COS function in mathematical functions and provides examples.

Syntax:

```
double cos(double number)
decimal cos(decimal number)
```

Description: This function calculates the cosine of number, which is a radian value.

Parameters: number, a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

Examples:

```
cos(3.1415926/2) = 2.6794896585028633e-8
cos(3.1415926) = -0.9999999999999986
```

6.6.1.8. COSH

This topic describes the COSH function in mathematical functions.

Syntax:

```
double cosh(double number)
decimal cosh(decimal number)
```

Description: This function calculates the hyperbolic cosine of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

6.6.1.9. COT

This topic describes the COT function in mathematical functions.

Syntax:

```
double cot(double number)
decimal cot(decimal number)
```

Description: This function calculates the cotangent of number, which is a radian value.

Parameters: number, a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

6.6.1.10. EXP

This topic describes the EXP function in mathematical functions.

Syntax:

```
double exp(double number)
decimal exp(decimal number)
```

Description: This function calculates the exponential value of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: The exponential value of number is returned. The value is of the DOUBLE or DECIMAL type. If the input value is NULL, NULL is returned.

6.6.1.11. FLOOR

This topic describes the FLOOR function in mathematical functions and provides examples.

Syntax:

```
bigint floor(double number)
bigint floor(decimal number)
```

Description: This function rounds down number and returns the nearest integer that is no greater than the value of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the BIGINT type is returned. If the input value is NULL, NULL is returned.

Examples:

```
floor(1.2) = 1
floor(1.9) = 1
floor(0.1) = 0
floor(-1.2) = -2
floor(-0.1) = -1
floor(0.0) = 0
floor(-0.0) = 0
```

6.6.1.12. LN

This topic describes the LN function in mathematical functions.

Syntax:

```
double ln(double number)
decimal ln(decimal number)
```

Description: This function calculates the natural logarithm of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned. If the input value is a negative value or 0, an error is returned.

Examples:

```
select ln(1000);
```

The following result is returned:

```
+-----+
| _c0   |
+-----+
| 6.907755278982137 |
+-----+
```

6.6.1.13. LOG

This topic describes the LOG function in mathematical functions.

Syntax:

```
double log(double base, double x)
decimal log(decimal base, decimal x)
```

Description: This function calculates the logarithm of x whose base number is $base$.

Parameters:

- **base**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- **x**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: The logarithm value of the DOUBLE or DECIMAL type is returned. If an input value is NULL, NULL is returned. If an input value is a negative value or 0, an error is returned. If the value of $base$ is 1, an error is returned. The value 1 causes division by zero.

Examples:

```
select log(10,100);
```

The following result is returned:

```
+-----+
| _c0   |
+-----+
| 2.0    |
+-----+
```

6.6.1.14. POW

This topic describes the POW function in mathematical functions.

Syntax:

```
double pow(double x, double y)
decimal pow(decimal x, decimal y)
```

Description: This function calculates the y th power of x , namely, x^y .

Parameters:

- **x**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an

error is returned.

- **y**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If an input value is NULL, NULL is returned.

Examples:

```
select pow(10,2);
```

The following result is returned:

```
+-----+
| _c0   |
+-----+
| 100.0 |
+-----+
```

6.6.1.15. RAND

This topic describes the RAND function in mathematical functions and provides examples.

Syntax:

```
double rand(bigint seed)
```

Description: This function returns a random number of the DOUBLE type based on seed. The value ranges from 0 to 1.

Parameters: The seed parameter supports a value of the BIGINT type. It specifies the starting point in generating random numbers. This parameter is optional.

Return value: A value of the DOUBLE type is returned.

Examples:

```
select rand();
select rand(1);
```

6.6.1.16. ROUND

This topic describes the ROUND function in mathematical functions and provides examples.

Syntax:

```
double round(double number, [bigint decimal_places])
decimal round(decimal number, [bigint decimal_places])
```

Description: This function returns a number rounded to the specified decimal place.

Parameters:

- **number**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- **decimal_places**: a constant of the BIGINT type. It specifies the decimal place to which the number is rounded. If the input value is of another data type, an error is returned. If this parameter is not specified, the number is

rounded to the ones place. The default value is 0.

Note `decimal_places` supports negative values. A negative value indicates counting from the decimal point to the left, and the decimal part is excluded. If `decimal_places` exceeds the length of the integer part, 0 is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If an input value is NULL, NULL is returned.

Examples:

```
round(125.315) = 125.0
round(125.315, 0) = 125.0
round(125.315, 1) = 125.3
round(125.315, 2) = 125.32
round(125.315, 3) = 125.315
round(-125.315, 2) = -125.32
round(123.345, -2) = 100.0
round(null) = null
round(123.345, 4) = 123.345
round(123.345, -4) = 0.0
```

6.6.1.17. SIN

This topic describes the SIN function in mathematical functions.

Syntax:

```
double sin(double number)
decimal sin(decimal number)
```

Description: This function calculates the sine of number, which is a radian value.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

6.6.1.18. SINH

This topic describes the SINH function in mathematical functions.

Syntax:

```
double sinh(double number)
decimal sinh(decimal number)
```

Description: This function calculates the hyperbolic sine of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

6.6.1.19. SQRT

This topic describes the SQRT function in mathematical functions.

Syntax:

```
double sqrt(double number)
decimal sqrt(decimal number)
```

Description: This function calculates the square root of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. It must be greater than 0. If it is less than 0, an error is returned. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

6.6.1.20. TAN

This topic describes the TAN function in mathematical functions.

Syntax:

```
double tan(double number)
decimal tan(decimal number)
```

Description: This function calculates the tangent of number, which is a radian value.

Parameters: number, a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

6.6.1.21. TANH

This topic describes the TANH function in mathematical functions.

Syntax:

```
double tanh(double number)
decimal tanh(decimal number)
```

Description: This function calculates the hyperbolic tangent of number.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

6.6.1.22. TRUNC

This topic describes the TRUNC function in mathematical functions and provides examples.

Syntax:

```
double trunc(double number[, bigint decimal_places])
decimal trunc(decimal number[, bigint decimal_places])
```

Description: This function truncates the input value of number to a specified number of decimal places.

Parameters:

- **number**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- **decimal_places**: the decimal place, which is a constant of the BIGINT type. This parameter indicates the position where the number is truncated. If it is of another data type, it is implicitly converted into the BIGINT type. If you do not specify this parameter, the number is truncated to the ones place.

Note `decimal_places` can be a negative value, which indicates that the number is truncated from the decimal point to the left and the decimal part is left out. If `decimal_places` exceeds the length of the integer part, 0 is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned. If an input value is NULL, NULL is returned.

Note

- If a value of the DOUBLE type is returned, the return value may not be properly displayed. This issue exists in all systems. For more information, see `trunc(125.815,1)` in the following example.
- The number is filled with zeros from the specified position.

Examples:

```
trunc(125.815) = 125.0
trunc(125.815, 0) =125.0
trunc(125.815, 1) = 125.8000000000000001
trunc(125.815, 2) = 125.81
trunc(125.815, 3) = 125.815
trunc(-125.815, 2) = -125.81
trunc(125.815, -1) = 120.0
trunc(125.815, -2) = 100.0
trunc(125.815, -3) = 0.0
trunc(123.345, 4) = 123.345
trunc(123.345, -4) = 0.0
```

6.6.1.23. Additional mathematical functions

6.6.1.23.1. Usage notes

This topic provides notes for you to use additional mathematical functions.

MaxCompute 2.0 provides additional mathematical functions. If you want to call additional mathematical functions, you must add the following SET statement before the SQL statement that contains the additional mathematical functions:

```
set odps.sql.type.system.odps2=true;
```

Note You must simultaneously submit and execute the SET statement and the SQL statement that contains the additional mathematical functions.

6.6.1.23.2. LOG2

This topic describes the LOG2 function in mathematical functions and provides examples.

Syntax:

```
double log2(double number)
double log2(decimal number)
```

Description: This function calculates the logarithm of number with the base number of 2.

Parameters: number, a value of the DOUBLE or DECIMAL type.

Return value: A value of the DOUBLE type is returned. If the input value is 0 or NULL, NULL is returned.

Examples:

```
log2(null) = null
log2(0) = null
log2(8) = 3.0
```

6.6.1.23.3. LOG10

This topic describes the LOG10 function in mathematical functions and provides examples.

Syntax:

```
double log10(double number)
double log10(decimal number)
```

Description: This function calculates the logarithm of number with the base number of 10.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type.

Return value: A value of the DOUBLE type is returned. If the input value is 0 or NULL, NULL is returned.

Examples:

```
log10(null)=null
log10(0)=null
log10(8)=0.9030899869919435
```

6.6.1.23.4. BIN

This topic describes the BIN function in mathematical functions and provides examples.

Syntax:

```
string bin(bigint number)
```

Description: This function calculates the binary code of number.

Parameters: The number parameter supports a value of the BIGINT type.

Return value: A value of the STRING type is returned. If the input value is 0, 0 is returned. If the input value is NULL, NULL is returned.

Examples:

```
bin(0) = '0'
bin(null) = 'null'
bin(12) = '1100'
```

6.6.1.23.5. HEX

This topic describes the HEX function in mathematical functions and provides examples.

Syntax:

```
string hex(bigint number)
string hex(string number)
string hex(binary number)
```

Description: This function converts an integer or a string into a hexadecimal number.

Parameters: If number is of the BIGINT type, a hexadecimal number is returned. If number is of the STRING type, a string in hexadecimal format is returned.

Return value: A value of the STRING type is returned. If the input value is 0, 0 is returned. If the input value is NULL, NULL is returned.

Examples:

```
hex(0) = '0'
hex('abc') = '616263'
hex(17) = '11'
hex('17') = '3137'
hex(null)
-- An exception occurs and the execution fails.
```

6.6.1.23.6. UNHEX

This topic describes the UNHEX function in mathematical functions and provides examples.

Syntax:

```
binary unhex(string number)
```

Description: This function converts a hexadecimal string into a string.

Parameters: The number parameter specifies a hexadecimal string.

Return value: A value of the BINARY type is returned. If the input value is 0, an error is returned. If the input value is NULL, NULL is returned.

Examples:

```
unhex('616263') = 'abc'
unhex(616263) = 'abc'
```

6.6.1.23.7. RADIANS

This topic describes the RADIANS function in mathematical functions and provides examples.

Syntax:

```
double radians(double number)
```

Description: This function converts a degree into a radian value.

Parameters: number, a value of the DOUBLE type.

Return value: A value of the DOUBLE type is returned. If the input value is NULL, NULL is returned.

Examples:

```
radians(90) = 1.5707963267948966
radians(0) = 0.0
radians(null) = null
```

6.6.1.23.8. DEGREES

This topic describes the DEGREES function in mathematical functions and provides examples.

Syntax:

```
double degrees(double number)
double degrees(decimal number)
```

Description: This function converts a radian value to a degree.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type.

Return value: A value of the DOUBLE type is returned. If the input value is NULL, NULL is returned.

Examples:

```
degrees(1.5707963267948966) = 90.0
degrees(0) = 0.0
degrees(null) = null
```

6.6.1.23.9. SIGN

This topic describes the SIGN function in mathematical functions and provides examples.

Syntax:

```
double sign(double number)
double sign(decimal number)
```

Description: This function returns the sign of an input value.

Parameters: The number parameter supports a value of the DOUBLE or DECIMAL type.

Return value: A value of the DOUBLE type is returned. If the input value is a positive value, 1.0 is returned. If the input value is a negative value, -1.0 is returned. If the input value is 0, 0.0 is returned. If the input value is NULL, NULL is returned.

Examples:

```
sign(-2.5) = -1.0
sign(2.5) = 1.0
sign(0) = 0.0
sign(null) = null
```

6.6.1.23.10. E

This topic describes the E function in mathematical functions and provides examples.

Syntax:

```
double e()
```

Description: This function returns the value of e.

Return value: A value of the DOUBLE type is returned.

Examples:

```
e() = 2.718281828459045
```

6.6.1.23.11. PI

This topic describes the PI function in mathematical functions and provides examples.

Syntax:

```
double pi()
```

Description: This function calculates the value of π .

Return value: A value of the DOUBLE type is returned.

Examples:

```
pi() = 3.141592653589793
```

6.6.1.23.12. FACTORIAL

This topic describes the FACTORIAL function in mathematical functions and provides examples.

Syntax:

```
bigint factorial(int number)
```

Description: This function calculates the factorial of number.

Parameters: number indicates a value of the INT type. The value ranges from 0 to 20.

Return value: A value of the BIGINT type is returned. If the input value is 0, 1 is returned. If the input value is NULL or a value that does not fall into the range from 0 to 20, NULL is returned.

Examples:

```
factorial(5) = 120 --5! = 5*4*3*2*1 = 120
```

6.6.1.23.13. CBRT

This topic describes the CBRT function in mathematical functions and provides examples.

Syntax:

```
double cbrt(double number)
```

Description: This function calculates the cube root of number.

Parameters: The number parameter supports a value of the DOUBLE type.

Return value: A value of the DOUBLE type is returned. If the input value is NULL, NULL is returned.

Examples:

```
cbirt(8) = 2
cbirt(null) = null
```

6.6.1.23.14. SHIFLEFT

This topic describes the SHIFLEFT function in mathematical functions and provides examples.

Syntax:

```
int shifleft(tinyint|smallint|int number1, int number2)
bigint shifleft(bigint number1, int number2)
```

Description: This function shifts a value left by a specific number of places (<<).

Parameters:

- number1: an integer of the TINYINT, SMALLINT, INT, or BIGINT type.
- number2: an integer of the INT type.

Return value: A value of the INT or BIGINT type is returned.

Examples:

```
shifleft(1,2)=4
-- Shift the binary value of 1 two places to the left (1<<2, 0001 shifted to 0100).
shifleft(4,3)=32
-- Shift the binary value of 4 three places to the left (4<<3, 0100 shifted to 100000).
```

6.6.1.23.15. SHIFTRIGHT

This topic describes the SHIFTRIGHT function in mathematical functions and provides examples.

Syntax:

```
int shiftright(tinyint|smallint|int number1, int number2)
bigint shiftright(bigint number1, int number2)
```

Description: This function shifts a value right by a specific number of places (>>).

Parameters:

- number1: an integer of the TINYINT, SMALLINT, INT, or BIGINT type.
- number2: an integer of the INT type.

Return value: A value of the INT or BIGINT type is returned.

Examples:

```
shiftright(4,2) = 1
-- Shift the binary value of 4 two places to the right (4>>2, 0100 shifted to 0001).
shiftright(32,3) = 4
-- Shift the binary value of 32 three places to the right (32>>3, 100000 shifted to 0100).
```

6.6.1.23.16. SHIFTRIGHTUNSIGNED

This topic describes the SHIFTRIGHTUNSIGNED function in mathematical functions and provides examples.

Syntax:

```
int shiftrightunsigned(tinyint|smallint|int number1, int number2)
bigint shiftrightunsigned(bigint number1, int number2)
```

Description: This function shifts an unsigned value right by a specific number of places (>>>).

Parameters:

- number1: an integer of the TINYINT, SMALLINT, INT, or BIGINT type.
- number2: an integer of the INT type.

Return value: A value of the INT or BIGINT type is returned.

Examples:

```
shiftrightunsigned(8,2) = 2
-- Shift the binary unsigned value of 8 two places to the right (8>>>2, 1000 shifted to be 0010).
shiftrightunsigned(-14,2) = 1073741820
-- Shift the binary value of -14 two places to the right (-14>>>2, 11111111 11111111 11111111 111100
10 shifted to be 00111111 11111111 11111111 11111100).
```

6.6.1.23.17. FORMAT_NUMBER

This topic describes the FORMAT_NUMBER function in mathematical functions and provides examples.

Syntax:

```
string format_number(float|double|decimal expr1, expr2)
```

Description: This function converts a number to a string in the specified format.

Parameters:

- expr1: a numeric expression that you want to format.
- expr2: the number of decimal places. It can be of the INT type. It can also be expressed in the format of #,###,###.##.

 **Note**

- If expr2 is greater than 0, the value is rounded to the specified place after the decimal point.
- If expr2 is equal to 0, the value has no decimal point or fractional part.
- If expr2 is less than 0 or greater than 340, an error is returned.

Return value: A value of the STRING type is returned.

Examples:

```
select format_number(5.230134523424545456,3);
-- The value 5.230 is returned.
select format_number(12332.123456, '#,###,###,###. ###');
-- The value 12,332.123 is returned.
```

6.6.1.23.18. WIDTH_BUCKET

This topic describes the WIDTH_BUCKET function in mathematical functions and provides examples.

Syntax:

```
WIDTH_BUCKET(NUMERIC expr, NUMERIC min_value, NUMERIC max_value, INT num_buckets)
```

Description: This function specifies the number of buckets and the minimum and maximum values of the acceptable range for a bucket. It allows you to construct equi-width buckets, in which the bucket range is divided into intervals that have an identical size. It returns the ID of the bucket into which the value of a specific expression falls. This function supports the following data types: DECIMAL(precision,scale) in the MaxCompute V2.0 data type edition, BIGINT, INT, FLOAT, DOUBLE, and DECIMAL.

Parameters:

- `expr`: the expression for which you want to identify the matching bucket ID.
- `min_value`: the minimum value of the acceptable range for the bucket.
- `max_value`: the maximum value of the acceptable range for the bucket. The value must be greater than `min_value`.
- `num_buckets`: the number of buckets. The value must be greater than 0.

Return value: A value of the BIGINT type is returned. The value ranges from 0 to `num_buckets` plus 1. If the value of `expr` is less than `min_value`, 0 is returned. If the value of `expr` is greater than `max_value`, the value of `num_buckets` plus 1 is returned. If the value of `expr` is NULL, NULL is returned. In other cases, the ID of the bucket into which the value falls is returned. The bucket ID is named based on the following formula: `Bucket ID =`

```
FLOOR[num_buckets × (expr - min_value)/(max_value - min_value) + 1] .
```

Examples:

```
SELECT key,value,WIDTH_BUCKET(value,100,500,5) as value_group
FROM VALUES
  (1,99),
  (2,100),
  (3,199),
  (4,200),
  (5,499),
  (6,500),
  (7,501),
  (8,NULL)
AS t(key,value);
```

The following result is returned:

```
+-----+-----+-----+
| key  | value | value_group |
+-----+-----+-----+
| 1    | 99    | 0           |
| 2    | 100   | 1           |
| 3    | 199   | 2           |
| 4    | 200   | 2           |
| 5    | 499   | 5           |
| 6    | 500   | 6           |
| 7    | 501   | 6           |
| 8    | \N    | \N         |
+-----+-----+-----+
```

6.6.2. String functions

6.6.2.1. CHAR_MATCHCOUNT

This topic describes the CHAR_MATCHCOUNT function in string functions and provides examples.

Syntax:

```
bigint char_matchcount(string str1, string str2)
```

Description: This function returns the number of characters that belong to str1 and appear in str2.

Parameters: The str1 and str2 parameters support a value of the STRING type. The values must be valid UTF-8 strings. If invalid characters are found during the comparison of the two strings, a negative value is returned.

Return value: A value of the BIGINT type is returned. If an input value is NULL, NULL is returned.

Examples:

```
char_matchcount('abd', 'aabc') = 2
-- The characters a and b of str1 appear in str2.
```

6.6.2.2. CHR

This topic describes the CHR function in string functions.

Syntax:

```
string chr(bigint ascii)
```

Description: This function converts specified an ASCII code into the required character.

Parameters: ascii indicates the ASCII code in the BIGINT type. If the input value is of the STRING, DOUBLE, or DECIMAL type, it is implicitly converted into a value of the BIGINT type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. The input value ranges from 0 to 255. If the input value does not fall into this range, an error is returned. If an input value is NULL, NULL is returned.

6.6.2.3. CONCAT

This topic describes the CONCAT function in string functions and provides examples.

Syntax:

```
string concat(string a, string b...)
```

Description: This function concatenates all specified strings.

Parameters: a and b, values of the STRING type. If an input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If an input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the parameters are not specified or an input value is NULL, NULL is returned.

Examples:

```
concat('ab', 'c') = 'abc'
concat() = null
concat('a', null, 'b') = null
```

6.6.2.4. INSTR

This topic describes the INSTR function in string functions and provides examples.

Syntax:

```
bigint instr(string str1, string str2[, bigint start_position[, bigint nth_appearance]])
```

Description: This function determines the position of substring str2 in string str1.

Parameters:

- **str1**: a value of the STRING type, which indicates the string in which you search for the substring. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.
- **str2**: a value of the STRING type, which indicates the substring you want to search for. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.
- **start_position**: a value of the BIGINT type. If the input value is of another data type, an error is returned. It indicates the character in str1 from which the search starts. The default value is 1, which indicates that the search starts from the first character.
- **nth_appearance**: a value of the BIGINT type. If the value is greater than 0, it indicates the position where the substring matches the string for the nth_appearance time. If the value is of another data type or less than or equal to 0, an error is returned.

Return value: A value of the BIGINT type is returned. If str2 is not matched in str1, 0 is returned. If an input value is NULL, NULL is returned. If str2 is an empty string, the substring always matches the string.

Examples:

```
instr('Tech on the net', 'e') = 2  
instr('Tech on the net', 'e', 1, 1) = 2  
instr('Tech on the net', 'e', 1, 2) = 11  
instr('Tech on the net', 'e', 1, 3) = 14
```

6.6.2.5. IS_ENCODING

This topic describes the IS_ENCODING function in string functions and provides examples.

Syntax:

```
boolean is_encoding(string str, string from_encoding, string to_encoding)
```

Description: This function determines whether the input string str can be converted from the character set specified by from_encoding to the character set specified by to_encoding. It can be used to determine whether the input string is garbled. Generally, from_encoding is set to utf-8, and to_encoding is set to gbk.

Parameters:

- **str**: a value of the STRING type. An empty string can belong to any character set.
- **from_encoding** and **to_encoding**: a value of the STRING type. The two parameters specify the source and destination character sets. If an input value is NULL, NULL is returned.

Return value: A value of the BOOLEAN type is returned. If str can be converted, true is returned. Otherwise, false is returned.

Examples:

```
is_encoding('test', 'utf-8', 'gbk') = true  
is_encoding('test', 'utf-8', 'gb2312') = true
```

6.6.2.6. KEYVALUE

This topic describes the KEYVALUE function in string functions and provides examples for reference only.

Syntax:

```
KEYVALUE (STRING srcStr, STRING split1, STRING split2, STRING key)
KEYVALUE (STRING srcStr, STRING key) //split1 = ";", split2 = ":"
```

Description: This function splits the source string srcStr into key-value pairs by split1, separates key-value pairs by split2, and then returns the value of the specified key.

Parameters:

- srcStr: the source string that you want to split.
- key: a value of the STRING type. After you split the source string by split1 and split2, the value of the specified key is returned.
- split1 and split2: the delimiters that are used to split strings. You must split the source string by using the specified delimiters. If you do not specify these two parameters, the default value of split1 is a semicolon (;) and that of split2 is a colon (:). If multiple key-value pairs are obtained after the source string is split by split1 and multiple delimiters specified by split2 are used to split the pairs, the returned result is undefined.

Return value: A value of the STRING type is returned. If the value of split1 or split2 is NULL, NULL is returned. If the value of srcStr or key is NULL, or key is not matched, NULL is returned. If multiple key-value pairs match the key, the value that corresponds to the first matched key is returned.

The following examples are provided:

Example 1:

```
KEYVALUE('0:1\;1:2', 1) = '2'
```

The source string is "0:1\;1:2". split1 and split2 use their default values. The default value of split1 is a semicolon (;) and that of split2 is a colon (:). After the source string is split by split1, the key-value pairs of 0:1\;1:2 are generated. After the key-value pairs are split by split2, the key-value pairs change to the following form:

```
0 1/
1 2
-- The value 2 that corresponds to key 1 is returned.
```

Example 2:

```
KEYVALUE("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;";", "\;";", ":", "tf") = "21910" value:21910
```

The source string is

"\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;".

After the source string is split by \; specified by split1, the following key-value pairs are generated:

```
decreaseStore:1, xcard:1, isB2C:1, tf:21910, cart:1, shipping:2, pf:0, market:shoes, instPayAmount:0
```

After the key-value pairs are split by a colon (:) specified by split2, the key-value pairs change to the following form:

```
decreaseStore 1
xcard 1
isB2C 1
tf 21910
cart 1
shipping 2
pf 0
market shoes
instPayAmount 0
-- The value 21910 that corresponds to key tf is returned.
```

6.6.2.7. LENGTH

This topic describes the LENGTH function in string functions and provides examples.

Syntax:

```
bigint length(string str)
```

Description: This function returns the length of the string str.

Parameters: The str parameter supports a value of the STRING type. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the BIGINT type is returned. If the input value is NULL, NULL is returned. If the input value is not encoded in UTF-8, -1 is returned.

Examples:

```
length('china') = 5
```

6.6.2.8. LENGTHB

This topic describes the LENGTHB function in string functions and provides examples.

Syntax:

```
bigint lengthb(string str)
```

Description: This function calculates the length of str in bytes.

Parameters: str, a value of the STRING type. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the BIGINT type is returned. If the input value is NULL, NULL is returned.

Examples:

```
lengthb('hi! China') = 10
```

6.6.2.9. MD5

This topic describes the MD5 function in string functions.

Syntax:

```
string md5(string value)
```

Description: This function calculates the MD5 value of value.

Parameters: value, a value of the STRING type. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

6.6.2.10. PARSE_URL

This topic describes the PARSE_URL function in string functions and provides examples.

Syntax:

```
string parse_url(string url, string part[,string key])
```

Description: This function parses url and extracts information by key.

Parameters:

- url: the URL to parse. If url is NULL, NULL is returned. If url is an invalid URL, an error is returned.
- part: a value of the STRING type. Valid values: HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO. The input value is not case-sensitive. If the input value is invalid, an error is returned.
- key: If part is QUERY, the value of key is extracted from the query string in URL and is returned. Otherwise, key is omitted.

Return value: A value of the STRING type is returned.

Examples:

```
url = file://username:password@example.com:8042/over/there/index.dtb? type=animal&name=narwhal#nose
parse_url('url', 'HOST') = "example.com"
parse_url('url', 'PATH') = "/over/there/index.dtb"
parse_url('url', 'QUERY') = "type=animal&name=narwhal"
parse_url('url', 'QUERY', 'name') = "narwhal"
parse_url('url', 'REF') = "nose"
parse_url('url', 'PROTOCOL') = "file"
parse_url('url', 'AUTHORITY') = "username:password@example.com:8042"
parse_url('url', 'FILE') = "/over/there/index.dtb? type=animal&name=narwhal"
parse_url('url', 'USERINFO') = "username:password"
```

6.6.2.11. REGEXP_EXTRACT

This topic describes the REGEXP_EXTRACT function in string functions and provides examples.

Syntax:

```
string regexp_extract(string source, string pattern[, bigint occurrence])
```

Description: This function splits source based on the regular expression specified by pattern and returns the characters in group at the nth occurrence, where n is specified by occurrence.

Parameters:

- source: a value of the STRING type. This parameter indicates the string that you want to split.
- pattern: a constant of the STRING type. If pattern is an empty string or group is not specified in pattern, an

error is returned.

- **occurrence**: a constant of the BIGINT type, which must be greater than or equal to 0. If the input value is of another data type or is smaller than 0, an error is returned. If you do not specify this parameter, the default value 1 is used. The value 1 indicates that characters in the first group are returned. If you set occurrence to 0, all substrings that match the regular expression specified by pattern are returned.

 **Note** Data is encoded by using UTF-8.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned.

Examples:

```
regexp_extract('foothebar', 'foo(. *) ( bar)', 1) = the
regexp_extract('foothebar', 'foo(. *) ( bar)', 2) = bar
regexp_extract('foothebar', 'foo(. *) ( bar)', 0) = foothebar
regexp_extract('8d99d8', '8d(\\d+)d8') = 99
-- If the regular expression is submitted on the MaxCompute client, use two backslashes (\\) as the
escape characters.
regexp_extract('foothebar', 'foothebar')
-- An error is returned because group is not specified in pattern.
```

6.6.2.12. REGEXP_INSTR

This topic describes the REGEXP_INSTR function in string functions and provides examples.

Syntax:

```
bigint regexp_instr(string source, string pattern[,bigint start_position[, bigint nth_occurrence[, b
igint return_option]])
```

Description: This function returns the start or end position of the substring that matches pattern at the nth occurrence, where n is specified by nth_occurrence, in the source string from the start position specified by start_position.

Parameters:

- **source**: a value of the STRING type. This parameter indicates the string that you want to search.
- **pattern**: a constant of the STRING type. If pattern is an empty string, an error is returned.
- **start_position**: a constant of the BIGINT type. This parameter indicates the start position for the search. If you do not specify this parameter, the default value 1 is used. If the value is of another data type or less than or equal to 0, an error is returned.
- **nth_occurrence**: a constant of the BIGINT type. If you do not specify this parameter, the default value 1 is used, indicating the position where a substring matches pattern in the search for the first time. If the value is of another data type or less than or equal to 0, an error is returned.
- **return_option**: a constant of the BIGINT type. Valid values are 0 and 1. If the value is of another data type or invalid, an error is returned. The value 0 indicates that the start position of the matched substring is returned. The value 1 indicates that the end position of the matched substring is returned.

Return value: A value of the BIGINT type is returned. This parameter indicates the start or end position of the matched substring returned in the source string based on the type specified by return_option. If an input value is NULL, NULL is returned.

Examples:

```
regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14
```

6.6.2.13. REGEXP_SUBSTR

This topic describes the REGEXP_SUBSTR function in string functions and provides examples.

Syntax:

```
string regexp_substr(string source, string pattern[, bigint start_position[, bigint nth_occurrence]]
)
```

Description: This function returns the substring that matches a given pattern at the nth occurrence, where n is specified by nth_occurrence, in the source string from the start position specified by start_position.

Parameters:

- source: a value of the STRING type. It specifies the string that contains the substring to return.
- pattern: a constant of the STRING type. It specifies the pattern used to match the substring. If the pattern value is NULL, an error is returned.
- start_position: a constant of the BIGINT type. It must be greater than 0. If it is of another data type or is less than or equal to 0, the function returns an error. If start_position is not specified, the default value 1 is used. In this case, the function starts matching from the first character of the source string.
- nth_occurrence: a constant of the BIGINT type. It must be greater than 0. If it is of another data type or is less than or equal to 0, an error is returned. If it is not specified, the default value 1 is used. In this case, the function returns the substring that first matches the specified pattern.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned. If no substrings match the specified pattern, NULL is returned.

Examples:

```
regexp_substr ("I love aliyun very much", "a[[:alpha:]]{5}") = "aliyun"
regexp_substr('I have 2 apples and 100 bucks!', '[:blank:][:alnum:]*', 1, 1) = " have"
regexp_substr('I have 2 apples and 100 bucks!', '[:blank:][:alnum:]*', 1, 2) = " 2"
```

6.6.2.14. REGEXP_COUNT

This topic describes the REGEXP_COUNT function in string functions and provides examples.

Syntax:

```
bigint regexp_count(string source, string pattern[, bigint start_position])
```

Description: This function returns the number of times a substring matches a given pattern in the source string from the start position specified by start_position.

Parameters:

- source: a value of the STRING type. It specifies the string that contains the substring to match. If the input value is of another data type, an error is returned.
- pattern: a constant of the STRING type. It specifies the pattern used to match the substring. If the pattern value is an empty string or is of another data type, an error is returned.
- start_position: a constant of the BIGINT type. It must be greater than 0. If it is of another data type or is less than or equal to 0, the function returns an error. If start_position is not specified, the default value 1 is used. In this case, the function starts matching from the first character of the source string.

Return value: A value of the BIGINT type is returned. If an input value is NULL, NULL is returned. If no substrings match the specified pattern, 0 is returned.

Examples:

```
regexp_count('abababc', 'a.c') = 1
regexp_count('abcde', '[:alpha:]{2}', 3) = 1
```

6.6.2.15. REGEXP_REPLACE

This topic describes the REGEXP_REPLACE function in string functions and provides examples.

Syntax:

```
string regexp_replace(string source, string pattern, string replace_string[, bigint occurrence])
```

Description: This function substitutes the string specified by `replace_string` for the substring that matches a given pattern at the `n`th occurrence, where `n` is specified by `occurrence`, in the source string, and returns the result.

Parameters:

- `source`: a value of the STRING type. It specifies the string that contains the substring to substitute.
- `pattern`: a constant of the STRING type. It specifies the pattern used to match the substring. If the value of `pattern` is NULL, an error is returned.
- `replace_string`: a value of the STRING type. It specifies the string to substitute for the substring that matches `pattern`.
- `occurrence`: a constant of the BIGINT type. It specifies the number of times at which the substring matches the pattern and is substituted with `replace_string`. It must be greater than or equal to 0. If the input value is 0, all matched substrings are substituted. If the input value is less than 0 or is not of the BIGINT type, an error is returned. Default value: 0.

 **Note** The action of referencing a character class that does not exist is undefined.

Return value: A value of the STRING type is returned. If the function references a character class that does not exist, substitution is not performed. If `replace_string` is NULL and substrings match `pattern`, NULL is returned. If `replace_string` is NULL and no substrings match `pattern`, the source string is returned. If an input value is NULL, NULL is returned.

Examples:

```
regexp_replace("123.456.7890", "([[:digit:]]{3})\\.[[:digit:]]{3}\\.[[:digit:]]{4}", "(\\1)\\2-\\3", 0) = "(123)456-7890"
regexp_replace("abcd", "(.)", "\\1 ", 0) = "a b c d "
regexp_replace("abcd", "(.)", "\\1 ", 1) = "a bcd"
regexp_replace("abcd", "(.)", "\\2", 1) = "abcd"
-- Only one character class is defined in pattern. The second referenced character class does not exist.
-- Try to avoid this action. The result of referencing a non-existent group is undefined.
regexp_replace("abcd", "(. *)\\.\\$", "\\2", 0) = "d"
regexp_replace("abcd", "a", "\\1", 0) = "bcd"
-- No character class is defined in pattern. \\1 references a non-existent character class.
-- Try to avoid this action. The result of referencing a non-existent group is undefined.
```

6.6.2.16. SPLIT_PART

This topic describes the SPLIT_PART function in string functions and provides examples.

Syntax:

```
string split_part(string str, string separator, bigint start[, bigint end])
```

Description: This function uses a delimiter specified by `separator` to split `str`, and returns a substring that starts from the character specified by `start` and ends with the character specified by `end`.

Parameters:

- `str`: a value of the `STRING` type, which indicates the string that you want to split. If the input value is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type, it is implicitly converted into a value of the `STRING` type before calculation. If the input value is of another data type, an error is returned.
- `separator`: a constant of the `STRING` type, which indicates the delimiter used to split a string. It can be a character or string. If it is of another data type, an error is returned.
- `start`: a constant of the `BIGINT` type, which must be greater than 0. If the value is not a constant or is of another data type, an error is returned. It indicates the start number of the segment to be returned. The number starts from 1. If `end` is not specified, the segment specified by `start` is returned.
- `end`: a constant of the `BIGINT` type. It must be greater than or equal to the value of `start`. Otherwise, an error is returned. It indicates the end number of the segment to be returned. If the value is not a constant or is of another data type, an error is returned. If this parameter is not specified, the last segment is returned.

Return value: A value of the `STRING` type is returned.

- If you set `start` to a value greater than the number of segments, an empty string is returned.
- If `separator` is absent in `str` and `start` is set to 1, the entire `str` is returned.
- If `str` is an empty string, an empty string is returned.
- If `separator` is an empty string, `str` is returned.
- If you set `end` to a value greater than the number of segments, all segments are returned.
- If an input value is `NULL`, `NULL` is returned.

Examples:

```
split_part('a,b,c,d', ',', 1) = 'a'  
split_part('a,b,c,d', ',', 1, 2) = 'a,b'  
split_part('a,b,c,d', ',', 10) = ''
```

6.6.2.17. SUBSTR

This topic describes the `SUBSTR` function in string functions and provides examples.

Syntax:

```
string substr(string str, bigint start_position[, bigint length])
```

Description: This function returns a substring that starts from `start_position` in `str` and has a length specified by `length`.

Parameters:

- `str`: a value of the `STRING` type. If the input value is of the `BIGINT`, `DECIMAL`, `DOUBLE`, or `DATETIME` type, it is implicitly converted to a value of the `STRING` type before calculation. If the input value is of another data type, an error is returned.
- `start_position`: a value of the `BIGINT` type. The function starts counting from the first character. If `start_position` is a negative value, the start position is counted backward from the last character of the string. The value -1 indicates the last character. If the input value is of another data type, an error is returned.
- `length`: a value of the `BIGINT` type. It specifies the length of the substring. The input value must be greater than 0. If the input value is of another data type or is less than or equal to 0, an error is returned.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned.

 **Note** If length is not specified, this function returns a substring that starts from start_position in str to the end of str.

Examples:

```
substr("abc", 2) = "bc"
substr("abc", 2, 1) = "b"
substr("abc",-2,2) = "bc"
substr("abc",-3) = "abc"
```

6.6.2.18. TOLOWER

This topic describes the TOLOWER function in string functions and provides examples.

Syntax:

```
string tolower(string source)
```

Description: This function converts the string source to lowercase letters.

Parameters: The source parameter supports a value of the STRING type. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

Examples:

```
tolower("aBcd") = "abcd"
tolower("HAHACd") = "hahacd"
```

6.6.2.19. TOUPPER

This topic describes the TOUPPER function in string functions and provides examples.

Syntax:

```
string toupper(string source)
```

Description: This function converts source into uppercase letters.

Parameters: source, a value of the STRING type. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

Examples:

```
toupper("aBcd") = "ABCD"
toupper("HahaCd") = "HAHACD"
```

6.6.2.20. TO_CHAR

This topic describes the TO_CHAR function in string functions and provides examples.

Syntax:

```
string to_char(boolean value)
string to_char(bigint value)
string to_char(double value)
string to_char(decimal value)
```

Description: This function converts data of the BOOLEAN, BIGINT, DECIMAL, or DOUBLE type into the STRING type.

Parameters: value, a value of the BOOLEAN, BIGINT, DECIMAL, or DOUBLE type. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

Examples:

```
to_char(123) = '123'
to_char(true) = 'TRUE'
to_char(1.23) = '1.23'
to_char(null) = 'null'
```

6.6.2.21. TRIM

This topic describes the TRIM function in string functions.

Syntax:

```
string trim(string str)
```

Description: This function eliminates the spaces on the left and right sides of the string str.

Parameters: The str parameter supports a value of the STRING type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted to a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, an error is returned.

6.6.2.22. LTRIM

This topic describes the LTRIM function in string functions and provides examples.

Syntax:

```
string ltrim(string str)
```

Description: This function removes the spaces on the left side of str.

Parameters: str, a value of the STRING type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

Examples:

```
select ltrim(' abc ');
-- The following result is returned:
+-----+
| _c0 |
+-----+
| abc |
+-----+
```

6.6.2.23. RTRIM

This topic describes the RTRIM function in string functions and provides examples.

Syntax:

```
string rtrim(string str)
```

Description: This function eliminates the spaces on the right side of the string str.

Parameters: The str parameter supports a value of the STRING type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted to a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

Examples:

```
select rtrim('a abc ');
-- The following result is returned:
+-----+
| _c0 |
+-----+
| a abc |
+-----+
```

6.6.2.24. REVERSE

This topic describes the REVERSE function in string functions and provides examples.

Syntax:

```
string reverse(string str)
```

Description: This function returns a string in reverse order.

Parameters: The str parameter supports a value of the STRING type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted to a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, an error is returned.

Examples:

```
select reverse('abcdefg');
-- The following result is returned:
+-----+
| _c0 |
+-----+
| gfdecba |
+-----+
```

6.6.2.25. SPACE

This topic describes the SPACE function in string functions and provides examples.

Syntax:

```
string space(bigint n)
```

Description: This function returns a string of a length specified by n.

Parameters: The n parameter supports a value of the BIGINT type. The value cannot exceed 2 MB. If this parameter is left empty, an error is returned.

Return value: A value of the STRING type is returned.

Examples:

```
select length(space(10));
-- The value 10 is returned.
select space(4000000000000);
-- An error is returned because the length exceeds 2 MB.
```

6.6.2.26. REPEAT

This topic describes the REPEAT function in string functions and provides examples.

Syntax:

```
string repeat(string str, bigint n)
```

Description: This function returns n duplicates of the string str.

Parameters:

- str: a value of the STRING type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted to a value of the STRING type before calculation. If the input value is of another data type, an error is returned.
- n: a value of the BIGINT type. The returned string cannot exceed 2 MB in length. If this parameter is left empty, an error is returned.

Return value: A value of the STRING type is returned.

Examples:

```
select repeat('abc',5);
-- abcabcabcabcabc is returned.
```

6.6.2.27. ASCII

This topic describes the ASCII function in string functions and provides examples.

Syntax:

```
bigint ascii(string str)
```

Description: This function returns the ASCII code of the first character in the string str.

Parameters: The str parameter supports a value of the STRING type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted to a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the BIGINT type is returned.

Examples:

```
select ascii('abcde');  
-- The value 97 is returned.
```

6.6.2.28. Additional string functions

6.6.2.28.1. Usage notes

This topic provides notes for you to use additional string functions.

MaxCompute V2.0 provides additional string functions. You must add the following flag before the SQL statement that contains the additional string functions:

```
set odps.sql.type.system.odps2=true;
```

 **Note** You must submit the flag along with the SQL statement that contains the additional string functions for execution.

6.6.2.28.2. CONCAT_WS

This topic describes the CONCAT_WS function in string functions and provides examples.

Syntax:

```
string concat_ws(string SEP, string a, string b...)  
string concat_ws(string SEP, array)
```

Description: This function concatenates all input strings in an array by using a specified delimiter.

Parameters:

- SEP: the delimiter of the STRING type. If this parameter is not specified, an error is returned.
- a, b...: a value of the STRING type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted to a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If no input values are specified or an input value is NULL, NULL is returned.

Examples:

```
concat_ws(':', 'name', 'hanmeimei')='name:hanmeimei'  
concat_ws(':', 'avg', null, '34')=null
```

6.6.2.28.3. LPAD

This topic describes the LPAD function in string functions and provides examples.

Syntax:

```
string lpad(string a, int len, string b)
```

Description: This function pads the left side of a with b based on the length specified by len.

Parameters:

- len: a value of the INT type.
- a: a value of the STRING type.
- b: a value of the STRING type.

Return value: A value of the STRING type is returned. If len is smaller than the number of characters in a, a is truncated from the left to obtain a string with the number of characters specified by len. If len is 0, no value is returned.

Examples:

```
lpad('abcdefgh', 10, '12')='12abcdefgh'  
lpad('abcdefgh', 5, '12')='abcde'  
lpad('abcdefgh', 0, '12')  
-- No value is returned.
```

6.6.2.28.4. RPAD

This topic describes the RPAD function in string functions and provides examples.

Syntax:

```
string rpad(string a, int len, string b)
```

Description: This function pads the right side of string a with string b until the new padded string has len characters.

Parameters:

- len: a value of the INT type.
- a: a value of the STRING type.
- b: a value of the STRING type.

Return value: A value of the STRING type is returned. If len is smaller than the number of characters in a, a is truncated from the left to obtain a string with len characters, and the obtained string is returned. If len is 0, NULL is returned.

Examples:

```
rpad('abcdefgh', 10, '12')='abcdefgh12'  
rpad('abcdefgh', 5, '12')='abcde'  
rpad('abcdefgh', 0, '12')  
-- NULL is returned.
```

6.6.2.28.5. REPLACE

This topic describes the REPLACE function in string functions and provides examples.

Syntax:

```
string replace(string a, string OLD, string NEW)
```

Description: This function substitutes the string NEW for the part of string a that is exactly the same as the string OLD, and returns the string a.

Parameters: All the parameters support a value of the STRING type.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned.

Examples:

```
replace('ababab','abab','12')='12ab'  
replace('ababab','cdf','123')='ababab'  
replace('123abab456ab',null,'abab')=null
```

6.6.2.28.6. SOUNDEX

This topic describes the SOUNDEX function in string functions and provides examples.

Syntax:

```
string soundex(string a)
```

Description: This function converts a normal string to a string of the soundex type.

Parameters: The a parameter supports a value of the STRING type.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

Examples:

```
soundex('hello') = 'H400'
```

6.6.2.28.7. SUBSTRING_INDEX

This topic describes the SUBSTRING_INDEX function in string functions and provides examples.

Syntax:

```
string substring_index(string a, string SEP, int count))
```

Description: This function truncates the string a to a substring from the first character to the nth delimiter, where n is specified by count. If count is a positive value, the string is truncated from left to right. Otherwise, the string is truncated from right to left.

Parameters:

- a: a value of the STRING type.
- SEP: a value of the STRING type.
- count: a value of the INT type.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned.

Examples:

```
substring_index('https://help.aliyun.com', '.', 2)='https://help.aliyun'  
substring_index('https://help.aliyun.com', '.', -2)='aliyun.com'  
substring_index('https://help.aliyun.com', null, 2)=null
```

6.6.2.28.8. TRANSLATE

This topic describes the TRANSLATE function in string functions and provides examples.

Syntax:

```
string translate(string|varchar str1, string|varchar str2, string|varchar str3)
```

Description: This function replaces the common substring of str1 and str2 with str3.

Parameters: The str parameter supports a value of the STRING or VARCHAR type. If the input value is of the BIGINT, DECIMAL, DOUBLE, or DATETIME type, it is implicitly converted to a value of the STRING type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned.

Examples:

```
translate('MaxComputer', 'puter', 'pute')='MaxCompute'  
translate('aaa', 'b', 'c')='aaa'  
translate('MaxComputer', 'puter', null)=null
```

6.6.2.28.9. URL_ENCODE

This topic describes the URL_ENCODE function in string functions and provides examples.

Syntax:

```
string url_encode(string input[, string encoding])
```

Description: This function encodes the input string in the application/x-www-form-urlencoded MIME format and returns the encoded string.

- All letters remain unchanged.
- Periods (.), hyphens (-), asterisks (*), and underscores (_) remain unchanged.
- Spaces are converted to plus signs (+).
- Other characters are converted to byte values based on encoding. Each byte value is then represented in the format of %xy, where xy is the hexadecimal representation of the character value.

Parameters:

- input: the string to encode.
- encoding: the encoding format. GBK and UTF-8 are supported. If you do not specify this parameter, the default value UTF-8 is used.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned.

Examples:

```
URL_ENCODE('Example for URL_ENCODE:// (fdsf)') = "%E7%A4%BA%E4%BE%8Bfor+URL_ENCODE%3A%2F%2F+%28fdsf%29"
URL_ENCODE('Example for URL_ENCODE:// dsf(fasfs)', 'GBK') = "Example+for+URL_ENCODE+%3A%2F%2F+dsf%28fasfs%29"
```

6.6.2.28.10. URL_DECODE

This topic describes the URL_DECODE function in string functions and provides examples.

Syntax:

```
string url_decode(string input[, string encoding])
```

Description: This function converts an input string from the application/x-www-form-urlencoded MIME format into a normal string. This is the inverse function of URL_ENCODE.

- All letters remain unchanged.
- Periods (.), hyphens (-), asterisks (*), and underscores (_) remain unchanged.
- Each plus sign (+) is converted into a space.
- The function first performs decoding based on the percent sign (%). The sequences that are in the %xy format are converted into byte values. Then, the function performs decoding based on the encoding format specified by the second parameter. Consecutive byte values are decoded to the required strings based on the format specified by encoding.
- Other characters remain unchanged.
- The return value of the function is a string encoded in UTF-8.

Parameters:

- input: the string that you want to decode.
- encoding: the specified encoding format. Valid values include GBK and UTF-8. If you do not specify this parameter, the default value UTF-8 is used.

Return value: A value of the STRING type is returned. If an input value is NULL, NULL is returned.

Examples:

```
URL_DECODE('%E7%A4%BA%E4%BE%8Bfor+URL_DECODE%3A%2F%2F+%28fdsf%29') = "Example for URL_DECODE:// (fdsf)"
URL_DECODE('Example+for+URL_DECODE+%3A%2F%2F+dsf%28fasfs%29', 'GBK') = "Example for URL_DECODE:// dsf(fasfs)"
```

6.6.2.28.11. JSON_TUPLE

This topic describes the JSON_TUPLE function in string functions and provides examples.

Syntax:

```
STRING JSON_TUPLE(STRING json, STRING key1, STRING key2, ...)
```

Description: This function extracts specific strings from a standard JSON string based on a set of input keys, such as key1 and key2.

Parameters:

- json: a value of the STRING type. It specifies a standard JSON string.
- key: a value of the STRING type. It describes the JSON path. You can enter multiple keys at a time. A key cannot

start with a dollar sign (\$).

Return value: A value of the STRING type is returned.

- If the JSON parameter is empty or invalid, NULL is returned.
- If the key parameter is empty or invalid, NULL is returned. If the key value does not exist in the JSON string, it is considered invalid.
- If the JSON parameter is valid and the key value exists in the JSON string, the required string is returned.

🔍 Note

- This function can parse JSON data that contains Chinese characters.
- This function can parse nested JSON data.
- This function can parse JSON data that contains nested arrays.
- This function parses a JSON string in the same way as the GET_JSON_OBJECT function for which `set odps.sql.udf.getjsonobj.new=true;` is added. To parse a JSON string multiple times, you must call the GET_JSON_OBJECT function multiple times. However, the JSON_TUPLE function allows you to enter multiple keys at a time and parse the JSON string only once. This improves parsing efficiency.
- JSON_TUPLE is a user-defined table-valued function (UDTF). To select other columns, you can use JSON_TUPLE with LATERAL VIEW.

Examples:

Prepare a school table that contains the following data:

```
Table: school
+-----+-----+
| Id      | json      |
+-----+-----+
| 1       | {
          "School name": "Hupan college",
          "Location": "Hangzhou",
          "SchoolRank": "00",
          "Class1": {
            "Student": [{
              "studentId": 1,
              "scoreRankIn3Year": [1, 2, [3, 2, 6]]
            }, {
              "studentId": 2,
              "scoreRankIn3Year": [2, 3, [4, 3, 1]]
            }
          ]
        }
      }
+-----+-----+
```

Extract JSON objects.

```
select json_tuple(school.json, "SchoolRank", "Class1") as (item0, item1) from school;
-- The preceding and following statements are equivalent.
select get_json_object(school.json, "$.SchoolRank") item0, get_json_object(school.json, "$.Class1") item1 from school;
```

The following result is returned:

```
+-----+-----+
| item0 | item1 |
+-----+-----+
| 00    | {"Student":[{"studentId":1,"scoreRankIn3Year":[1,2,[3,2,6]]}, {"studentId":2,"scoreRankIn3Year": [2,3,[4,3,1]]}] } |
+-----+-----+
```

Parse JSON data that contains Chinese characters.

```
select json_tuple(school.json,"School name","Location") as (item0,item1) from school;
```

The following result is returned:

```
+-----+-----+
| item0 | item1 |
+-----+-----+
| Hupan college | Hangzhou |
+-----+-----+
```

Parse nested JSON data.

```
select sc.Id, q.item0, q.item1
from school sc LATERAL VIEW json_tuple(sc.json,"Class1.Student.[*].studentId","Class1.Student.[0].scoreRankIn3Year") q as item0,item1;
```

The following result is returned:

```
+-----+-----+-----+
| id      | item0 | item1 |
+-----+-----+-----+
| 1       | [1,2] | [1,2,[3,2,6]] |
+-----+-----+-----+
```

Parse JSON data that contains nested arrays.

```
select sc.Id, q.item0, q.item1
from school sc LATERAL VIEW json_tuple(sc.json,"Class1.Student[0].scoreRankIn3Year[2]","Class1.Student[0].scoreRankIn3Year[2][1]") q as item0,item1;
```

The following result is returned:

```
+-----+-----+-----+
| id      | item0 | item1 |
+-----+-----+-----+
| 1       | [3,2,6] | 2 |
+-----+-----+-----+
```

6.6.2.28.12. FROM_JSON

This topic describes the FROM_JSON function in string functions and provides examples.

Syntax:

```
FROM_JSON(jsonStr, schema)
```

Description: This function processes a JSON string and returns a value of the ARRAY, MAP, or STRUCT type with the specified schema.

Parameters:

- jsonStr: a value of the STRING type. This parameter specifies the JSON string to process.
- schema: the data structure. The syntax of schema is the same as that of table creation statements. Examples:
 - `array<bigint>`
 - `map<string, array<string>>`
 - `struct<a:int, b:double, c:map<string,string>>`

For the STRUCT type, you can also specify a value in the `'a INT, b DOUBLE'` format, which is equivalent to `struct<a:int, b:double>`.

The following table describes the mapping relationships between JSON and MaxCompute data types.

JSON	MaxCompute
Object	STRUCT / MAP / STRING
Array	ARRAY / STRING
Number	TINYINT / SMALLINT / INT / BIGINT / FLOAT / DOUBLE / DECIMAL / STRING
true/false	BOOLEAN / STRING
String	STRING
null	All types

Note For the OBJECT and ARRAY types, this function preferentially parses the data. For the types that are not supported by MaxCompute, this function ignores the data.

Return value: A value of the ARRAY, MAP, or STRUCT type is returned.

Note If the schema parameter is left empty, an error is returned.

Examples:

```
SELECT from_json('{"a":1, "b":0.8}', 'a INT, b DOUBLE');
-- {"a":1, "b":0.8} is returned.
SELECT from_json('{"time":"26/08/2015"}', 'time string');
-- {"time":"26/08/2015"} is returned.
SELECT from_json('{"a":1, "b":0.8}', 'a INT, b DOUBLE, c STRING');
-- {"a":1, "b":0.8, c: NULL} is returned.
SELECT from_json('[1, 2, 3, "a"]', 'array<BIGINT>');
-- [1, 2, 3] is returned.
```

6.6.2.28.13. TO_JSON

This topic describes the TO_JSON function in string functions and provides examples.

Syntax:

```
TO_JSON(expr)
```

Description: This function returns a JSON string for a value of a given type.

Parameters: The expr parameter supports a value of the ARRAY, MAP, or STRUCT type.

Return value: A value of the STRING type is returned.

Examples:

```
SELECT to_json(named_struct('a', 1, 'b', 2));
-- {"a":1,"b":2} is returned.
SELECT to_json(named_struct('time', "26/08/2015"));
-- {"time":"26/08/2015"} is returned.
SELECT to_json(array(named_struct('a', 1, 'b', 2)));
-- [{"a":1,"b":2}] is returned.
SELECT to_json(map('a', named_struct('b', 1)));
-- {"a":{"b":1}} is returned.
SELECT to_json(map('a', 1));
-- {"a":1} is returned.
SELECT to_json(array((map('a', 1))));
-- [{"a":1}] is returned.
```

6.6.3. Date functions

6.6.3.1. DATEADD

This topic describes the date function DATEADD and provides examples.

Syntax:

```
datetime dateadd(datetime date, bigint delta, string datepart)
```

Description: This function modifies a date value based on datepart and delta that you specified.

Parameters:

- **date:** a date value of the DATE, DATETIME, or TIMESTAMP type. If the input value is of the STRING type, it is implicitly converted into a value of the DATETIME type before the calculation. If the input value is of another data type, an error is returned.
- **delta:** a value of the BIGINT type, which indicates the interval to add to the specified part of the date value. If the input value is of the STRING or DOUBLE type, it is implicitly converted into a value of the BIGINT type before the calculation. If the input value is of another data type, an error is returned. If the value of delta is greater than 0, this function adds the interval to the date value. In other cases, this function subtracts the interval from the date value.

Note

- If you add or subtract the interval specified by delta at a date part, a carry or return at more significant date parts may occur. The year, month, hour, minute, and second parts are computed by using different numeral systems. The year part uses the base-10 numeral system. The month part uses the base-12 numeral system. The hour part uses the base-24 numeral system. The minute and second parts use the base-60 numeral system.
- If the DATEADD function adds an interval specified by delta to the month part of a date value of the DATETIME type and this operation does not cause an overflow of day, keep day unchanged. If the operation causes an overflow of day, set day to the last day of the specified month.

- **datepart:** the part you want to modify in the date value. The value is a constant of the STRING type. If the value is in an invalid format or is not a constant of the STRING type, an error is returned. The value of this parameter is specified in compliance with the rules of conversions between the STRING and DATETIME types.

The value `yyyy` indicates that the `DATEADD` function adds an interval to the year part of the date value. The value `mm` indicates that the `DATEADD` function adds an interval to the month part of the date value. This parameter also supports extended date formats, such as `-year`, `-month`, `-mon`, `-day`, and `-hour`. For more information about the rules of data type conversions, see [Conversion between string and datetime types](#).

Return value: A value of the `DATETIME` type is returned. If an input value is `NULL`, `NULL` is returned.

Examples:

- Example 1: common usage

```
select dateadd(datetime '2005-02-28 00:00:00', 1, 'dd') ;
-- The return value is 2005-03-01 00:00:00. After one day is added, the result is beyond the last
day of February. The actual date value is the first day of March.
select dateadd(datetime '2005-02-28 00:00:00', -1, 'dd');
-- The return value is 2005-02-27 00:00:00. One day is subtracted.
select dateadd(datetime '2005-02-28 00:00:00', 20, 'mm');
-- The return value is 2006-10-28 00:00:00. After 20 months are added, the month overflows, and th
e year increases by 1.
select dateadd(datetime '2005-02-28 00:00:00', 1, 'mm');
-- The return value is 2005-03-28 00:00:00. One month is added.
select dateadd(datetime '2005-01-29 00:00:00', 1, 'mm');
-- The return value is 2005-02-28 00:00:00. February in 2005 has only 28 days. Therefore, the last
day of February is returned.
select dateadd(datetime '2005-03-30 00:00:00', -1, 'mm');
-- The return value is 2005-02-28 00:00:00. One month is subtracted. February in 2005 has only 28
days. Therefore, the last day of February is returned.
```

- Example 2: usage of `DATEADD` in which a value of the `DATETIME` type is expressed as a constant

```
select dateadd(2005-03-30 00:00:00, -1, 'mm');
-- In MaxCompute SQL statements, a value of the DATETIME type cannot be directly expressed as a co
nstant. This statement uses an invalid expression for a value of the DATETIME type.
```

```
select dateadd(cast("2005-03-30 00:00:00" as datetime), -1, 'mm');
-- This statement uses the DATEADD function in which a constant of the STRING type is explicitly c
onverted to a value of the DATETIME type.
```

6.6.3.2. DATEDIFF

This topic describes the date function `DATEDIFF` and provides examples.

Syntax:

```
bigint datediff(datetime date1, datetime date2, string datepart)
```

Description: This function calculates the difference between `date1` and `date2`. The difference is measured in the time unit specified by `datepart`.

Parameters:

- `date1` and `date2`: the minuend and subtrahend. The values are of the `DATE`, `DATETIME`, or `TIMESTAMP` type. If an input value is of the `STRING` type, it is implicitly converted into a value of the `DATETIME` type before the calculation. If the input value is of another data type, an error is returned.
- `datepart`: the time unit, which is a constant of the `STRING` type. This parameter supports extended date formats. If `datepart` is not in the specified format or is of another data type, an error is returned.

Return value: A value of the `BIGINT` type is returned. If an input value is `NULL`, `NULL` is returned. If `date1` is earlier than `date2`, a negative value is returned.

Note

- If the difference between two dates is more precise than the unit specified by datepart, the excessive parts are discarded in the return value. For example, if the unit specified by datepart is day, the hour, minute, and second parts are discarded in the return value.
- This function omits the parts with smaller units based on the unit specified by datepart and calculates the result. The parts with smaller units refer to the excessive parts previously described.

Examples:

```
-- The start time is 2005-12-31 23:59:59 and the end time is 2006-01-01 00:00:00.
datediff(end, start, 'dd') = 1
datediff(end, start, 'mm') = 1
datediff(end, start, 'yyyy') = 1
datediff(end, start, 'hh') = 1
datediff(end, start, 'mi') = 1
datediff(end, start, 'ss') = 1
datediff(datetime'2013-05-31 13:00:00', '2013-05-31 12:30:00', 'ss') = 1800
datediff(datetime'2013-05-31 13:00:00', '2013-05-31 12:30:00', 'mi') = 30
-- The start time is 2018-06-04 19:33:23.234 and the end time is 2018-06-04 19:33:23.250. Date values with milliseconds do not adopt the standard DATETIME type and therefore cannot be implicitly converted into the DATETIME type. In this case, an explicit conversion is required.
datediff(to_date('2018-06-04 19:33:23.250', 'yyyy-MM-dd hh:mi:ss.ff3'),to_date('2018-06-04 19:33:23.234', 'yyyy-MM-dd hh:mi:ss.ff3') , 'ff3') = 16
```

6.6.3.3. DATEPART

This topic describes the date function DATEPART and provides examples.

Syntax:

```
bigint datepart(datetime date, string datepart)
```

Description: This function returns a specific part of a date value. The part is specified by datepart.

Parameters:

- **date:** a date value of the DATE, DATETIME, or TIMESTAMP type. If the input value is of the STRING type, it is implicitly converted into a value of the DATETIME type before the calculation. If the input value is of another data type, an error is returned.
- **datepart:** a constant of the STRING type. This parameter supports extended date formats. If datepart is not in the specified format or is of another data type, an error is returned.

Return value: A value of the BIGINT type is returned. If an input value is NULL, NULL is returned.

Examples:

```
datepart('2017-06-08 01:10:00', 'yyyy') = 2017
datepart('2017-06-08 01:10:00', 'mm') = 6
```

6.6.3.4. DATETRUNC

This topic describes the date function DATETRUNC and provides examples.

Syntax:

```
datetime datetrunc (datetime date,string datepart)
```

Description: This function truncates a date value to the accuracy specified by datepart and returns a new date.

Parameters:

- **date**: a date value of the DATE, DATETIME, or TIMESTAMP type. If the input value is of the STRING type, it is implicitly converted into a value of the DATETIME type before the calculation. If the input value is of another data type, an error is returned.
- **datepart**: a constant of the STRING type. This parameter supports extended date formats. If datepart is not in the specified format or is of another data type, an error is returned.

Return value: A value of the DATETIME type is returned. If an input value is NULL, NULL is returned.

Examples:

```
datetrunc('2017-12-07 16:28:46', 'yyyy') = 2017-01-01 00:00:00
-- The function truncates the date value to the accuracy of the year 2017 and returns the date value
that is accurate to the year. In this example, the date value that is accurate to the year 2017 is r
eturned.
datetrunc('2017-12-07 16:28:46', 'month') = 2017-12-01 00:00:00
-- The function truncates the date value to the accuracy of the month December and returns the date
value that is accurate to the month. In this example, the date value that is accurate to the month "
2017-12" is returned.
datetrunc('2017-12-07 16:28:46', 'DD') = 2017-12-07 00:00:00
-- The function truncates the date value to the accuracy of the day 07 and returns the date value th
at is accurate to the day. In this example, the date value that is accurate to the day "2017-12-07"
is returned.
```

6.6.3.5. GETDATE

This topic describes the GETDATE function in date functions.

Syntax:

```
datetime getdate()
```

Description: This function returns the current system time as a date value. MaxCompute uses UTC+8 as the standard time zone.

Return value: A value of the DATETIME type is returned. The value indicates the current date and time.

 **Note** In a MaxCompute SQL task that is executed in distributed mode, the GETDATE function always returns a fixed value. The return value is an arbitrary time during the execution of the MaxCompute SQL task. The time is accurate to seconds. In MaxCompute V2.0 that supports more data types, the time is accurate to milliseconds.

6.6.3.6. ISDATE

This topic describes the ISDATE function in date functions.

Syntax:

```
boolean isdate(string date, string format)
```

Description: This function determines whether a date string can be converted into a date value in a specified format. If the date string can be converted into a date value in the specified format, True is returned. Otherwise, False is returned.

Parameters:

- **date**: a value of the STRING type. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type, an error is returned.
- **format**: a constant of the STRING type. This parameter does not support extended date formats. If the input value is of another data type, an error is returned. If redundant format strings exist in format, this function converts the date string that corresponds to the first format string into a date value. The rest strings are considered delimiters. For example, `isdate("1234-yyyy", "yyyy-yyyy")` returns True.

Return value: A value of the BOOLEAN type is returned. If an input value is NULL, NULL is returned.

6.6.3.7. LASTDAY

This topic describes the LASTDAY function in date functions.

Syntax:

```
datetime lastday(datetime date)
```

Description: This function returns the last day of the month in which the date value falls. The value is accurate to days. The hour, minute, and second parts are expressed as 00:00:00.

Parameters: The date parameter supports a value of the DATETIME type. If the input value is of the STRING type, it is implicitly converted into a value of the DATETIME type before calculation. If the input value is of another data type, an error is returned.

Return value: A value of the DATETIME type is returned. If the input value is NULL, NULL is returned.

6.6.3.8. TO_DATE

This topic describes the TO_DATE function and provides examples of using this function.

Syntax:

```
datetime to_date(string date, string format)
```

Description: This function converts a date string in a specified format into a date value.

Parameters:

- **date**: a date value of the STRING type, which indicates the date string you want to convert. If the input value is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type, it is implicitly converted into a value of the STRING type before calculation. If the input value is of another data type or an empty string, an error is returned.
- **format**: a constant of the STRING type, which indicates a date format. If the input value is not a constant or is not of the STRING type, an error is returned. The Extended Date/Time Format (EDTF) is not supported. In this format, the characters are parsed as invalid characters and omitted. The value of this parameter must contain yyyy. Otherwise, an error is returned. If redundant format strings are included in the value of this parameter, only the date value that corresponds to the first format string is used, and the date values that correspond to the other format strings are parsed as delimiters. For example, `to_date('1234-2234 ', 'yyyy-yyyy')` is parsed as 1234-01-01 00:00:00.

Return value: A value of the DATETIME type is returned and the value is in the format of yyyy-mm-dd hh:mi:ss:ff3. If the value of any input parameter is NULL, NULL is returned.

Note In the format, yyyy indicates a 4-digit year, mm indicates a 2-digit month, dd indicates a 2-digit day, hh indicates an hour in 24-hour display, mi indicates a 2-digit minute, ss indicates a 2-digit second, and ff3 indicates a 3-digit millisecond.

Examples:

```
to_date('Alibaba2017-12*03', 'Alibabayyyy-mm*dd') = 2017-12-03 00:00:00
to_date('20170718', 'yyyymmdd') = 2017-07-18 00:00:00
to_date('201707182030', 'yyyymmddhhmi')=2017-07-18 20:30:00
to_date('2017718', 'yyyymmdd')
-- Invalid format. NULL is returned.
to_date('Alibaba2017-12*3', 'Alibabayyyy-mm*dd')
-- Invalid format. NULL is returned.
to_date('2017-24-01', 'yyyy')
-- Invalid format. NULL is returned.
to_date('20181030 15-13-12.345', 'yyyymmdd hh-mi-ss.ff3')=2018-10-30 15:13:12
```

6.6.3.9. TO_CHAR

This topic describes the TO_CHAR function and provides examples of using this function.

Syntax:

```
string to_char(datetime date, string format)
```

Description: This function converts a date value of the DATETIME type into a string in a specified format.

Parameters:

- **date**: a date value of the DATETIME type, which indicates the date value you want to convert. If the input value is of the STRING type, it is implicitly converted into the DATETIME type before calculation. If the input value is not of the STRING type, an error is returned.
- **format**: a constant of the STRING type. If it is not a constant or is not of the STRING type, an error is returned. In the format parameter, the date format part is replaced by the related data and other characters remain unchanged in the output.

Return value: A value of the STRING type is returned. If the value of an input parameter is NULL, NULL is returned.

Examples:

```
to_char('2017-12-03 00:00:00', 'Alibaba Cloud Financial Servicesyyyy-mm*dd') = 'Alibaba Cloud Financial Services2017-12*03'
to_char('2017-07-18 00:00:00', 'yyyymmdd') = '20170718'
to_char('Alibaba2017-12*3', 'Alibabayyyy-mm*dd')
-- NULL is returned.
to_char('2017-24-01', 'yyyy')
-- NULL is returned.
to_char('2017718', 'yyyymmdd')
-- NULL is returned.
```

6.6.3.10. UNIX_TIMESTAMP

This topic describes the UNIX_TIMESTAMP function and provides examples of using this function.

Syntax:

```
bigint unix_timestamp(datetime date)
```

Description: This function converts a date value to a UNIX timestamp that is an integer.

Parameters: date: a date value of the DATETIME type. If the input value is of the STRING type, it is implicitly converted into a value of the DATETIME type before calculation. If the input value is not of the STRING type, an error is returned. If you enable new data types, the implicit conversion fails. In this case, you must use the CAST function for conversion, for example, `unix_timestamp(cast(... as datetime))`.

Return value: A UNIX timestamp of the BIGINT type is returned. If the value of the input parameter is NULL, NULL is returned.

Examples:

```
select unix_timestamp(datetime'2009-03-20 11:11:00');
-- 1237518660 is returned.
```

6.6.3.11. FROM_UNIXTIME

This topic describes the FROM_UNIXTIME function and provides examples of using this function.

Syntax:

```
datetime from_unixtime(bigint unixtime)
```

Description: This function converts unixtime of the BIGINT type to a date value of the DATETIME type.

Parameters: unixtime: a date value of the BIGINT type in the UNIX format. Its value is accurate to seconds. If the input value is of the STRING, DOUBLE, or DECIMAL type, it is implicitly converted into a value of the BIGINT type before calculation.

Return value: A value of the DATETIME type is returned. If an input value is NULL, NULL is returned.

 **Note** In the Hive-compatible mode where `set odps.sql.hive.compatible=true;` has been run, if the value of the input parameter is of the STRING type, a date value of the STRING type is returned.

Examples:

```
from_unixtime(123456789) = 1973-11-30 05:33:09;
```

6.6.3.12. WEEKDAY

This topic describes the WEEKDAY function.

Syntax:

```
bigint weekday (datetime date)
```

Description: This function returns the day of the week for a specified date.

Parameters: date: a date value of the DATETIME type. If the input value is of the STRING type, it is implicitly converted into a value of the DATETIME type before calculation. If the input value is not of the STRING type, an error is returned.

Return value: A value of the BIGINT type is returned. If the value of the input parameter is NULL, NULL is returned. Monday is treated as the first day of a week and its return value is 0. Days are numbered in ascending order starting from 0. The return value for Sunday is 6.

6.6.3.13. WEEKOFYEAR

This topic describes the WEEKOFYEAR function and provides examples of using this function.

Syntax:

```
bigint weekofyear(datetime date)
```

Description: This function returns the calendar week of the year that the specified date falls in. Monday is treated as the first day of a week.

Note If a week spans two years, whether this week belongs to the previous year or the next year is based on which year contains more than four days. If more days fall in the previous year, the week is considered as the last week of the previous year. If more days fall in the next year, the week is considered as the first week of the next year.

Parameters: date: a date value of the DATETIME type. If the value of the input parameter is of the STRING type, it is implicitly converted into a value of the DATETIME type before calculation. If the value of the input parameter is not of the STRING type, an error is returned.

Return value: A value of the BIGINT type is returned. If the value of the input parameter is NULL, NULL is returned.

Examples:

```
select weekofyear(to_date("20141229", "yyyymmdd"));
-- The following result is returned:
+-----+
| _c0    |
+-----+
| 1      |
+-----+
-- 20141229 is in year 2014, but most days of the week fall in year 2015. Therefore, the return value 1 indicates the first week of year 2015.
select weekofyear(to_date("20141231", "yyyymmdd"));
-- 1 is returned.
select weekofyear(to_date("20151229", "yyyymmdd"));
-- 53 is returned.
```

6.6.3.14. Additional date functions

6.6.3.14.1. Usage notes

This topic describes the configuration operation that you must perform before you use the date functions that are added to MaxCompute V2.0.

MaxCompute V2.0 provides more date functions. You must add the following SET statement before the SQL statements for these new functions:

```
set odps.sql.type.system.odps2=true;
```

Note You must submit and execute the SET statement and the SQL statements for the new functions at the same time.

6.6.3.14.2. YEAR

This topic describes the YEAR function and provides examples of using this function.

Syntax:

```
INT YEAR(DATETIME/STRING date)
```

Description: This function returns the year in which the specified date value falls.

Parameters: date: a date value of the DATETIME or STRING type. The format of the date value must include yyyy-mm-dd and exclude redundant strings. Otherwise, NULL is returned.

Return value: A value of the INT type is returned.

Examples:

```
SELECT YEAR('1970-01-01 12:30:00');  
-- 1970 is returned.  
SELECT YEAR('1970-01-01');  
-- 1970 is returned.  
SELECT YEAR('70-01-01');  
-- 70 is returned.  
SELECT YEAR('1970-01-01');  
-- 1970 is returned.  
SELECT YEAR('1970/03/09');  
-- NULL is returned.  
SELECT YEAR(null);  
-- An error is returned.
```

6.6.3.14.3. QUARTER

This topic describes the QUARTER function and provides examples of using this function.

Syntax:

```
INT QUARTER (DATETIME/TIMESTAMP/STRING date)
```

Description: This function returns the quarter of the year for a date value. The quarter is an integer from 1 to 4.

Parameters: date: a date value of the DATETIME, TIMESTAMP, or STRING type. The format of the date value must include yyyy-mm-dd. If the date value is not of the DATETIME, TIMESTAMP, or STRING type, NULL is returned.

Return value: A value of the INT type is returned. If the value of an input parameter is NULL, NULL is returned.

Examples:

```
SELECT QUARTER('1970-11-12 10:00:00');  
-- 4 is returned.  
SELECT QUARTER('1970-11-12');  
-- 4 is returned.
```

6.6.3.14.4. MONTH

This topic describes the MONTH function and provides examples of using this function.

Syntax:

```
INT MONTH(DATETIME/STRING date)
```

Description: This function returns the month part of a date value.

Parameters: date: a value of the DATETIME or STRING type. The date format must include yyyy-mm-dd. If it is not of the DATETIME or STRING type, an error is returned.

Return value: A value of the INT type is returned.

Examples:

```
SELECT MONTH('2014-09-01');
-- 9 is returned.
SELECT MONTH('20140901');
-- NULL is returned.
```

6.6.3.14.5. DAY

This topic describes the DAY function and provides examples of using this function.

Syntax:

```
INT DAY(DATETIME/STRING date)
```

Description: This function returns the day of a specified date value.

Parameters: date: a date value of the DATETIME or STRING type. The format of the date value can be yyyy-mm-dd or yyyy-mm-dd hh:mi:ss. If the date value is not of the DATETIME or STRING type, an error is returned.

Return value: A value of the INT type is returned.

Examples:

```
SELECT DAY('2014-09-01');
-- 1 is returned.
SELECT DAY('20140901');
-- NULL is returned.
```

6.6.3.14.6. DAYOFMONTH

This topic describes the DAYOFMONTH function and provides examples of using this function.

Syntax:

```
int dayofmonth(date)
```

Description: This function returns the value of the day part of a date value. For example, if the date is October 13, 2019, 13 is returned after you run the `int dayofmonth(2019-10-13)` command.

Parameters: date: a date value of the STRING type. The date format must include yyyy-mm-dd. If it is not of the STRING type, an error is returned.

Return value: A value of the INT type is returned.

Examples:

```
dayofmonth('2019-09-01');
-- 1 is returned.
dayofmonth('20190901');
-- NULL is returned.
```

6.6.3.14.7. HOUR

This topic describes the HOUR function and provides examples of using this function.

Syntax:

```
INT HOUR(DATETIME/STRING date)
```

Description: This function returns the value of the hour part of a date.

Parameters: date: a value of the DATETIME or STRING type. If the value is not of the DATETIME or STRING type, an error is returned.

Return value: A value of the INT type is returned.

Examples:

```
SELECT HOUR('2014-09-01 12:00:00');  
-- 12 is returned.  
SELECT HOUR('12:00:00');  
-- 12 is returned.  
SELECT HOUR('20140901120000');  
-- NULL is returned.
```

6.6.3.14.8. MINUTE

This topic describes the MINUTE function and provides examples of using this function.

Syntax:

```
INT MINUTE(DATETIME/STRING date)
```

Description: This function returns the value of the minute part of a date.

Parameters: date: a value of the DATETIME or STRING type. If the value is not of the DATETIME or STRING type, an error is returned.

Return value: A value of the INT type is returned.

Examples:

```
SELECT MINUTE('2014-09-01 12:30:00');  
-- 30 is returned.  
SELECT MINUTE('12:30:00');  
-- 30 is returned.  
SELECT MINUTE('20140901120000');  
-- NULL is returned.
```

6.6.3.14.9. SECOND

This topic describes the SECOND function and provides examples of using this function.

Syntax:

```
INT SECOND(DATETIME/STRING date)
```

Description: This function returns the value of the second part of a date value.

Parameters: date: a date value of the DATETIME or STRING type. If it is not of the DATETIME or STRING type, an error is returned.

Return value: A value of the INT type is returned.

Examples:

```
SELECT SECOND('2014-09-01 12:30:45');
-- 45 is returned.
SELECT SECOND('12:30:45');
-- 45 is returned.
SELECT SECOND('20140901123045');
-- NULL is returned.
```

6.6.3.14.10. FROM_UTC_TIMESTAMP

This topic describes the FROM_UTC_TIMESTAMP function and provides examples of using this function.

Syntax:

```
timestamp from_utc_timestamp({any primitive type}*, string timezone)
```

Description: This function converts a UTC timestamp to a timestamp for a specified time zone.

Parameters:

- {any primitive type}*: the timestamp, which is a value of the TIMESTAMP, DATETIME, TINYINT, SMALLINT, INT, or BIGINT type. If the value is of the TINYINT, SMALLINT, INT, or BIGINT type, the unit is milliseconds.
- timezone: the time zone to which you want to convert the timestamp, such as Pacific Standard Time (PST).

Return value: A value of the TIMESTAMP type is returned.

Examples:

```
SELECT from_utc_timestamp(1501557840000, 'PST');
-- The unit of the input value is milliseconds and 2017-08-01 04:24:00 is returned.
SELECT from_utc_timestamp('1970-01-30 16:00:00', 'PST');
-- 1970-01-30 08:00:00.0 is returned.
SELECT from_utc_timestamp('1970-01-30', 'PST');
-- 1970-01-29 16:00:00.0 is returned.
```

6.6.3.14.11. CURRENT_TIMESTAMP

This topic describes the CURRENT_TIMESTAMP function and provides examples of using this function.

Syntax:

```
timestamp current_timestamp()
```

Description: This function returns the current timestamp. The return value is not fixed.

Return value: A value of the TIMESTAMP type is returned.

Examples:

```
select current_timestamp();
-- '2017-08-03 11:50:30.661' is returned.
```

6.6.3.14.12. ADD_MONTHS

This topic describes the ADD_MONTHS function and provides examples of using this function.

Syntax:

```
string add_months(string startdate, int nummonths)
```

Description: This function adds a specified number of months to a date specified by startdate. The number of months is specified by nummonths.

Parameters:

- startdate: a date value of the STRING type. The date format must contain yyyy-mm-dd. Otherwise, NULL is returned.
- num_months: a value of the INT type.

Return value: A value of the STRING type is returned and the value is in the format of yyyy-mm-dd.

Examples:

```
add_months('2017-02-14',3) = '2017-05-14'  
add_months('17-2-14',3) = '0017-05-14'  
add_months('2017-02-14 21:30:00',3) = '2017-05-14'  
add_months('20170214',3) = null
```

6.6.3.14.13. LAST_DAY

This topic describes the LAST_DAY function and provides examples of using this function.

Syntax:

```
string last_day(string date)
```

Description: This function returns the last date of the month part of a date value.

Parameters: date: a date value of the STRING type and in the format of yyyy-mm-dd hh:mi:ss or yyyy-mm-dd.

Return value: A value of the STRING type is returned and the value is in the format of yyyy-mm-dd.

Examples:

```
last_day('2017-03-04') = '2017-03-31'  
last_day('2017-07-04 11:40:00') = '2017-07-31'  
last_day('20170304') = null
```

6.6.3.14.14. NEXT_DAY

This topic describes the NEXT_DAY function and provides examples of using this function.

Syntax:

```
string next_day(string startdate, string week)
```

Description: This function returns the date of the first day that is later than startdate and matches the week value. The return value indicates the date of the specified day in the next week.

Parameters:

- startdate: a date value of the STRING type and in the format of yyyy-mm-dd hh:mi:ss or yyyy-mm-dd.
- week: a date value of the STRING type. The full name of a day in a week, or the first two or three letters of

the day, for example, MO, TUE, or FRIDAY.

Return value: A value of the STRING type is returned and the value is in the format of yyyy-mm-dd.

Examples:

```
next_day('2017-08-01','TU') = '2017-08-08'
next_day('2017-08-01 23:34:00','TU') = '2017-08-08'
next_day('20170801','TU') = null
```

6.6.3.14.15. MONTHS_BETWEEN

This topic describes the MONTHS_BETWEEN function and provides examples of using this function.

Syntax:

```
double months_between(datetime/timestamp/string date1, datetime/timestamp/string date2)
```

Description: This function returns the number of months between date1 and date2.

Parameters:

- date1: a value of the DATETIME, TIMESTAMP, or STRING type. The value is in the format of yyyy-mm-dd hh:mi:ss or yyyy-mm-dd.
- date2: a value of the DATETIME, TIMESTAMP, or STRING type. The value is in the format of yyyy-mm-dd hh:mi:ss or yyyy-mm-dd.

Return value: A value of the DOUBLE type is returned.

- If date1 is later than date2, a positive value is returned. If date2 is later than date1, a negative value is returned.
- If both date1 and date2 correspond to the last days of two months, the return value is an integer that represents the number of months. Otherwise, the return value is computed by using the following formula: $(date1 - date2)/31$

Examples:

```
months_between('1997-02-28 10:30:00', '1996-10-30') = 3.9495967741935485
months_between('1996-10-30', '1997-02-28 10:30:00') = -3.9495967741935485
months_between('1996-09-30', '1996-12-31') = -3.0
```

6.6.3.14.16. EXTRACT

This topic describes the EXTRACT function and provides examples of using this function.

Syntax:

```
int extract(<datepart> from <timestamp>)
```

Description: This function extracts the part specified by datepart from the time specified by timestamp.

Parameters:

- datepart: a value that can be set to YEAR, MONTH, DAY, HOUR, or MINUTE.
- timestamp: a date value of the TIMESTAMP type.

Return value: A value of the INT type is returned.

Examples:

```

set odps.sql.type.system.odps2=true;
select extract(year from '2019-05-01 11:21:00') year
      ,extract(month from '2019-05-01 11:21:00') month
      ,extract(day from '2019-05-01 11:21:00') day
      ,extract(hour from '2019-05-01 11:21:00') hour
      ,extract(minute from '2019-05-01 11:21:00') minute;
-- The following result is returned:
+-----+-----+-----+-----+-----+
| year | month | day  | hour | minute |
+-----+-----+-----+-----+
| 2019 | 5     | 1    | 11   | 21     |
+-----+-----+-----+-----+

```

If the time value specified in the SQL statement is invalid or exceeds the specified range, the return value is the remainder obtained by dividing the specified time value by the maximum value in the time range.

Examples:

```

set odps.sql.type.system.odps2=true;
select extract(hour from '2019-05-01 31:01:01') hour
      ,extract(minute from '2019-05-01 23:61:01') minute;
-- The following result is returned:
+-----+-----+
| hour | minute |
+-----+-----+
| 7    | 1      |
+-----+-----+
-- The maximum value of the hour part is 24, and the specified time value is 31. The return value is
7 (31/24).
-- The maximum value of the minute part is 60, and the specified time value is 61. The return value
is 1 (61/60).

```

6.6.4. Window functions

6.6.4.1. Overview

In MaxCompute SQL, window functions can be used for flexible data analysis and processing. This topic describes the precautions for using window functions and the syntax of window functions.

Notice

- Window functions can only be included in the SELECT clause.
- A window function cannot contain nested window functions or aggregate functions.
- Window functions cannot be used with aggregate functions of the same level.
- A maximum of five window functions can be used in a MaxCompute SQL statement.

Syntax:

```

window_func() over (partition by [col1,col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] windowing_clause)

```

Parameters:

- **PARTITION BY:** the columns in the window that is used for calculation. Rows with the same values of the partition key columns are considered in the same window. A window can contain a maximum of 100 million

rows of data. We recommend that the number of rows in a window not exceed 5 million. If the number of rows exceeds 5 million, an error is returned.

- **ORDER BY:** specifies how to sort data in a window.
- **windowing_clause:** You can use ROWS to specify how to define the window for the function. Valid values:
 - **rows between x preceding|following and y preceding|following:** indicates a window that ranges from the xth row preceding or following the current row to the yth row preceding or following the current row.
 - **rows x preceding|following:** indicates a window that ranges from the xth row preceding or following the current row to the current row.

 **Note**

- x and y must be integer constants greater than or equal to 0. Their values range from 0 to 10000. 0 indicates the current row.
- You must specify ORDER BY before you use ROWS to specify a window range.
- Window functions that allow you to use rows to define the window include AVG, COUNT, MAX, MIN, STDDEV, and SUM.

6.6.4.2. COUNT

This topic describes the COUNT function and provides examples of using this function.

Syntax:

```
Bigint count([distinct] expr) over(partition by [col1, col2...]  
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

Description: This function returns the counted values of specified rows.

Parameters:

- **expr:** a value of any data type. If the value for a row is NULL, this row is not used for calculation. If the distinct keyword is specified, only the distinct values are counted.
- **partition by [col1, col2...]:** the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]:** If ORDER BY is not specified, the counted value of expr in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the value accumulated from the starting row to the current row in the current window is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be used.

Return value: A value of the BIGINT type is returned.

Examples:

The test_src table contains the user_id column of the BIGINT type.

```

select user_id,count(user_id) over (partition by user_id) as count from test_src;
+-----+-----+
| user_id | count  |
+-----+-----+
| 1      | 3      |
| 1      | 3      |
| 1      | 3      |
| 2      | 1      |
| 3      | 1      |
+-----+-----+
-- If ORDER BY is not specified, the counted value of the user_id column in the current window is re
turned.
select user_id,count(user_id) over (partition by user_id order by user_id) as count from test_src;
+-----+-----+
| user_id | count  |
+-----+-----+
| 1      | 1      |      -- This row is the starting row of this window.
| 1      | 2      |      -- Two rows are found from the starting row to the current row. 2 is r
eturned.
| 1      | 3      |
| 2      | 1      |
| 3      | 1      |
+-----+-----+
-- If ORDER BY is specified, the counted value from the starting row to the current row in the curre
nt window is returned.

```

If duplicate values are specified for ORDER BY, the processing method is based on the compatibility between MaxCompute and Hive.

- If MaxCompute is not compatible with Hive, the value of COUNT for each row is returned.

```

set odps.sql.hive.compatible=false;
select user_id, price, count(price) over
(partition by user_id order by price) as count from test_src;
+-----+-----+-----+
| user_id | price  | count |
+-----+-----+-----+
| 1      | 4.5    | 1     | -- This row is the starting row of this window.
| 1      | 5.5    | 2     | -- The value of COUNT for the second row is 2.
| 1      | 5.5    | 3     | -- The value of COUNT for the third row is 3.
| 1      | 6.5    | 4     |
| 2      | NULL   | 0     |
| 2      | 3.0    | 1     |
| 3      | NULL   | 0     |
| 3      | 4.0    | 1     |
+-----+-----+-----+

```

- If MaxCompute is compatible with Hive, the return value is the value of COUNT for the last row of the rows with the same value.

```

set odps.sql.hive.compatible=true;
select user_id, price, count(price) over
  (partition by user_id order by price) as count from test_src;
+-----+-----+-----+
| user_id | price | count |
+-----+-----+-----+
| 1       | 4.5   | 1     | -- This row is the starting row of this window.
| 1       | 5.5   | 3     | -- The value of COUNT for the second row is the same as t
he value of COUNT for the third row because the prices of the two rows are the same.
| 1       | 5.5   | 3     | -- The value of COUNT for the third row is 3.
| 1       | 6.5   | 4     |
| 2       | NULL  | 0     |
| 2       | 3.0   | 1     |
| 3       | NULL  | 0     |
| 3       | 4.0   | 1     |
+-----+-----+-----+

```

6.6.4.3. AVG

This topic describes the AVG function and provides examples of using this function.

Syntax:

```

avg([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])

```

Description: This function returns the average value of input values in specified rows.

Parameters:

- **expr**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If the input value is not of the STRING or BIGINT type, an error is returned. If the input value for a row is NULL, the row is not used for calculation. Values of the BOOLEAN type are not used for calculation. If the distinct keyword is specified, the average value of distinct values is calculated.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]**: If ORDER BY is not specified, the average value of all values in the current window is returned. If ORDER BY is specified, the returned results are sorted in a specified order and the accumulated average value of the values from the starting row to the current row is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be used.

Return value: A value of the DOUBLE type is returned.

If duplicate values are specified for ORDER BY, the processing method is based on the compatibility between MaxCompute and Hive.

- If MaxCompute is not compatible with Hive, the return value is the average value for each row.

```

set odps.sql.hive.compatible=false;
select user_id, price, avg(price) over
  (partition by user_id order by price) from test_src;
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1       | 4.5   | 4.5 | -- This row is the starting row of this window.
| 1       | 5.5   | 5.0 | -- The return value is the average value of the
values for the first and second rows.
| 1       | 5.5   | 5.166666666666667 | -- The return value is the average value of the
values from the first row to the third row.
| 1       | 6.5   | 5.5 |
| 2       | NULL  | NULL |
| 2       | 3.0   | 3.0 |
| 3       | NULL  | NULL |
| 3       | 4.0   | 4.0 |
+-----+-----+-----+

```

- If MaxCompute is compatible with Hive, the return value is the average value of values for the last row of the rows with the same value.

```

set odps.sql.hive.compatible=true;
select user_id, price, avg(price) over
  (partition by user_id order by price) from test_src;
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1       | 4.5   | 4.5 | -- This row is the starting row of this window.
| 1       | 5.5   | 5.166666666666667 | -- The return value is the average value of value
s from the first row to the third row.
| 1       | 5.5   | 5.166666666666667 | -- The return value is the average value of value
s from the first row to the third row.
| 1       | 6.5   | 5.5 |
| 2       | NULL  | NULL |
| 2       | 3.0   | 3.0 |
| 3       | NULL  | NULL |
| 3       | 4.0   | 4.0 |
+-----+-----+-----+

```

6.6.4.4. MAX

This topic describes the MAX function.

Syntax:

```

max([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])

```

Description: This function calculates the maximum value of specified rows in the current window.

Parameters:

- expr: a value of any data type except BOOLEAN. If the value for a row is NULL, this row is not used for calculation. If the distinct keyword is specified, the maximum value of distinct values is calculated. The calculation result is not affected regardless of whether the parameter is specified.
- partition by [col1, col2...]: the columns in the window that is used for calculation.
- order by [col1[asc|desc], col2[asc|desc]: If ORDER BY is not specified, the maximum value of all values in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and

the maximum value of values from the starting row to the current row in the current window is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be used.

Return value: A value of the same data type as expr is returned.

6.6.4.5. MIN

This topic describes the MIN function.

Syntax:

```
min([distinct] expr) over(partition by [col1, col2...]  
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

Description: This function returns the minimum value of values in specified rows.

Parameters:

- **expr**: a value of any data type except BOOLEAN. If the value for a row is NULL, the row is not used for calculation. If the distinct keyword is specified, the minimum value of distinct values is calculated. The calculation result is not affected regardless of whether the parameter is specified.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]**: If ORDER BY is not specified, the minimum value in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the minimum value of values from the starting row to the current row in the current window is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be used.

Return value: A value of the same data type as expr is returned.

6.6.4.6. MEDIAN

This topic describes the MEDIAN function.

Syntax:

```
double median(double number1,number2...) over(partition by [col1, col2...])  
decimal median(decimal number1,number2...) over(partition by [col1, col2...])
```

Description: This function calculates the median.

Parameters:

- **number1,number2...:** numbers of the DOUBLE or DECIMAL type. You can enter 1 to 255 numbers. At least one number is required. If the input value is of the DOUBLE type, it is automatically converted into an array of the DOUBLE type before calculation. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If it is not of the STRING or BIGINT type, an error is returned. If the input value is NULL, NULL is returned.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.

Return value: A value of the DOUBLE or DECIMAL type is returned.

6.6.4.7. STDDEV

This topic describes the STDDEV function and provides examples of using this function.

Syntax:

```

Double stddev([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
Decimal stddev([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])

```

Description: This function calculates the population standard deviation of values in specified rows.

Parameters:

- **expr**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If the input value is not of the STRING or BIGINT type, an error is returned. If the value for a row is NULL, this row is not used for calculation. If the distinct keyword is specified, the population standard deviation of distinct values is calculated.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]]**: If ORDER BY is not specified, the population standard deviation of rows in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the population standard deviation from the starting row to the current row in the current window is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be used.

Return value: If the input value is of the DECIMAL type, a value of the DECIMAL type is returned. Otherwise, a value of the DOUBLE type is returned.

Examples:

```
select window, seq, stddev_pop('1\01') over (partition by window order by seq) from dual;
```

If duplicate values are specified for ORDER BY, the processing method is based on the compatibility between MaxCompute and Hive.

- If MaxCompute is not compatible with Hive, the return value is the value of STDDEV for each row.

```

set odps.sql.hive.compatible=false;
select user_id, price, stddev(price) over
(partition by user_id order by price) from test_src;
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1       | 4.5   | 0.0 | -- This row is the starting row of this window.
| 1       | 5.5   | 0.5 | -- The return value is the value of STDDEV for t
he first and second rows.
| 1       | 5.5   | 0.4714045207910316 | -- The return value is the value of STDDEV from
the first row to the third row.
| 1       | 6.5   | 0.7071067811865475 |
| 2       | NULL  | NULL |
| 2       | 3.0   | 0.0 |
| 3       | NULL  | NULL |
| 3       | 4.0   | 0.0 |
+-----+-----+-----+

```

- If MaxCompute is compatible with Hive, the return value is the value of STDDEV for the last row of the rows with the same value.

```

set odps.sql.hive.compatible=true;
select user_id, price, stddev(price) over
  (partition by user_id order by price) from test_src;
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1       | 4.5   | 0.0 | -- This row is the starting row of this window.
| 1       | 5.5   | 0.4714045207910316 | -- The return value is the value of STDDEV from t
he first row to the third row.
| 1       | 5.5   | 0.4714045207910316 | -- The return value is the value of STDDEV from t
he first row to the third row.
| 1       | 6.5   | 0.7071067811865475 |
| 2       | NULL  | NULL |
| 2       | 3.0   | 0.0 |
| 3       | NULL  | NULL |
| 3       | 4.0   | 0.0 |
+-----+-----+-----+

```

6.6.4.8. STDDEV_SAMP

This topic describes the STDDEV_SAMP function.

Syntax:

```

Double stddev_samp([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
Decimal stddev_samp([distinct] expr) over((partition by [col1,col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])

```

Description: This function calculates the sample standard deviation.

Parameters:

- **expr**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If the input value is not of the STRING or BIGINT type, an error is returned. If the value for a row is NULL, the row is not used for calculation. If the distinct keyword is specified, the sample standard deviation of distinct values is calculated.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]**: If ORDER BY is not specified, the sample standard deviation in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the sample standard deviation from the starting row to the current row in the current window is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be used.

Return value: If the input value is of the DECIMAL type, a value of the DECIMAL type is returned. Otherwise, a value of the DOUBLE type is returned.

6.6.4.9. SUM

This topic describes the SUM function and provides examples of using this function.

Syntax:

```

sum([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])

```

Description: This function returns the sum of values in specified rows.

Parameters:

- `expr`: a value of the `DOUBLE`, `DECIMAL`, or `BIGINT` type. If the input value is of the `STRING` type, it is implicitly converted to the `DOUBLE` type before calculation. If it is not of the `STRING` type, an error is returned. If the value for a row is `NULL`, the row is not used for calculation. If the `distinct` keyword is specified, the sum of distinct values is calculated.
- `partition by [col1, col2...]`: the columns in the window that is used for calculation.
- `order by [col1[asc|desc], col2[asc|desc]]`: If `ORDER BY` is not specified, the sum of the `expr` values in the current window is returned. If `ORDER BY` is specified, the returned results are sorted in the specified order and the sum of the values from the starting row to the current row in the current window is returned.

 **Note** If the `distinct` keyword is specified, `ORDER BY` cannot be used.

Return value: If the input value is of the `BIGINT` type, a value of the `BIGINT` type is returned. If the input value is of the `DECIMAL` type, a value of the `DECIMAL` type is returned. If the input value is of the `DOUBLE` or `STRING` type, a value of the `DOUBLE` type is returned.

If duplicate values are specified for `ORDER BY`, the processing method is based on the compatibility between MaxCompute and Hive.

- If MaxCompute is not compatible with Hive, the return value is the sum of values for each row.

```
set odps.sql.hive.compatible=false;
select user_id, price, sum(price) over
  (partition by user_id order by price) from test_src;
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1       | 4.5   | 4.5 | -- This row is the starting row of this window.
| 1       | 5.5   | 10.0| -- The return value is the sum of values for the first a
nd second rows.
| 1       | 5.5   | 15.5| -- The return value is the sum of values from the first
row to the third row.
| 1       | 6.5   | 22.0|
| 2       | NULL  | NULL|
| 2       | 3.0   | 3.0 |
| 3       | NULL  | NULL|
| 3       | 4.0   | 4.0 |
+-----+-----+-----+
```

- If MaxCompute is compatible with Hive, the return value is the sum of values for the last row of the rows with the same value.

```

set odps.sql.hive.compatible=true;
select user_id, price, sum(price) over
  (partition by user_id order by price) from test_src;
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1       | 4.5   | 4.5 | -- This row is the starting row of this window.
| 1       | 5.5   | 15.5| -- The return value is the sum of values from the first
row to the third row.
| 1       | 5.5   | 15.5| -- The return value is the sum of values from the first
row to the third row.
| 1       | 6.5   | 22.0|
| 2       | NULL  | NULL|
| 2       | 3.0   | 3.0 |
| 3       | NULL  | NULL|
| 3       | 4.0   | 4.0 |
+-----+-----+-----+

```

6.6.4.10. DENSE_RANK

This topic describes the DENSE_RANK function and provides examples of using this function.

Syntax:

```

Bigint dense_rank() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])

```

Description: This function returns the ranking of values. The data in rows with the same column value in ORDER BY has the same ranking.

Parameters:

- partition by [col1, col2...]: the columns in the window that is used for calculation.
- order by [col1[asc|desc], col2[asc|desc]: the value on which the ranking values are based.

Return value: A value of the BIGINT type is returned.

Examples:

The emp table contains the following data:

```

| empno | ename | job | mgr | hiredate | sal | comm | deptno |
7369, SMITH, CLERK, 7902, 1980-12-17 00:00:00, 800, , 20
7499, ALLEN, SALESMAN, 7698, 1981-02-20 00:00:00, 1600, 300, 30
7521, WARD, SALESMAN, 7698, 1981-02-22 00:00:00, 1250, 500, 30
7566, JONES, MANAGER, 7839, 1981-04-02 00:00:00, 2975, , 20
7654, MARTIN, SALESMAN, 7698, 1981-09-28 00:00:00, 1250, 1400, 30
7698, BLAKE, MANAGER, 7839, 1981-05-01 00:00:00, 2850, , 30
7782, CLARK, MANAGER, 7839, 1981-06-09 00:00:00, 2450, , 10
7788, SCOTT, ANALYST, 7566, 1987-04-19 00:00:00, 3000, , 20
7839, KING, PRESIDENT, , 1981-11-17 00:00:00, 5000, , 10
7844, TURNER, SALESMAN, 7698, 1981-09-08 00:00:00, 1500, 0, 30
7876, ADAMS, CLERK, 7788, 1987-05-23 00:00:00, 1100, , 20
7900, JAMES, CLERK, 7698, 1981-12-03 00:00:00, 950, , 30
7902, FORD, ANALYST, 7566, 1981-12-03 00:00:00, 3000, , 20
7934, MILLER, CLERK, 7782, 1982-01-23 00:00:00, 1300, , 10
7948, JACCKA, CLERK, 7782, 1981-04-12 00:00:00, 5000, , 10
7956, WELAN, CLERK, 7649, 1982-07-20 00:00:00, 2450, , 10
7956, TEBAGE, CLERK, 7748, 1982-12-30 00:00:00, 1300, , 10

```

Group employees by department, sort the employees in each group in descending order of sal, and then obtain the rankings of employees in each group.

```
select deptno
       , ename
       , sal
       , dense_rank() over (partition by deptno order by sal desc) as nums
-- DEPTNO (department) is the column in the window that is used for calculation. Values in the sal c
column are sorted to generate the ranking of each employee.
from emp;
-- The following result is returned:
```

deptno	ename	sal	nums
10	JACCKA	5000.0	1
10	KING	5000.0	1
10	CLARK	2450.0	2
10	WELAN	2450.0	2
10	TEBAGE	1300.0	3
10	MILLER	1300.0	3
20	SCOTT	3000.0	1
20	FORD	3000.0	1
20	JONES	2975.0	2
20	ADAMS	1100.0	3
20	SMITH	800.0	4
30	BLAKE	2850.0	1
30	ALLEN	1600.0	2
30	TURNER	1500.0	3
30	MARTIN	1250.0	4
30	WARD	1250.0	4
30	JAMES	950.0	5

6.6.4.11. RANK

This topic describes the RANK function and provides examples of using this function.

Syntax:

```
Bigint rank() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])
```

Description: This function determines the ranking of values in rows with the same column value, after the values are sorted in descending order by using the ORDER BY clause.

Parameters:

- partition by [col1, col2...]: the columns in the window that is used for calculation.
- order by [col1[asc|desc], col2[asc|desc]: the field based on which values are ranked.

Return value: A value of the BIGINT type is returned.

Examples:

The emp table contains the following data:

```

| empno | ename | job | mgr | hiredate | sal | comm | deptno |
7369, SMITH, CLERK, 7902, 1980-12-17 00:00:00, 800, , 20
7499, ALLEN, SALESMAN, 7698, 1981-02-20 00:00:00, 1600, 300, 30
7521, WARD, SALESMAN, 7698, 1981-02-22 00:00:00, 1250, 500, 30
7566, JONES, MANAGER, 7839, 1981-04-02 00:00:00, 2975, , 20
7654, MARTIN, SALESMAN, 7698, 1981-09-28 00:00:00, 1250, 1400, 30
7698, BLAKE, MANAGER, 7839, 1981-05-01 00:00:00, 2850, , 30
7782, CLARK, MANAGER, 7839, 1981-06-09 00:00:00, 2450, , 10
7788, SCOTT, ANALYST, 7566, 1987-04-19 00:00:00, 3000, , 20
7839, KING, PRESIDENT, , 1981-11-17 00:00:00, 5000, , 10
7844, TURNER, SALESMAN, 7698, 1981-09-08 00:00:00, 1500, 0, 30
7876, ADAMS, CLERK, 7788, 1987-05-23 00:00:00, 1100, , 20
7900, JAMES, CLERK, 7698, 1981-12-03 00:00:00, 950, , 30
7902, FORD, ANALYST, 7566, 1981-12-03 00:00:00, 3000, , 20
7934, MILLER, CLERK, 7782, 1982-01-23 00:00:00, 1300, , 10
7948, JACCKA, CLERK, 7782, 1981-04-12 00:00:00, 5000, , 10
7956, WELAN, CLERK, 7649, 1982-07-20 00:00:00, 2450, , 10
7956, TEBAGE, CLERK, 7748, 1982-12-30 00:00:00, 1300, , 10

```

Group employees by department, sort the employees in each group in descending order of sal, and then obtain the ranking values of employees in each group.

```

select deptno
       , ename
       , sal
       , rank() over (partition by deptno order by sal desc) as nums
-- DEPTNO (department) is the column in the window that is used for calculation. Values in the sal c
column are sorted to generate the ranking value for each employee.
from emp;
-- The following result is returned:
+-----+-----+-----+-----+
| deptno | ename | sal      | nums |
+-----+-----+-----+-----+
| 10     | JACCKA | 5000.0   | 1    |
| 10     | KING   | 5000.0   | 1    |
| 10     | CLARK  | 2450.0   | 3    |
| 10     | WELAN  | 2450.0   | 3    |
| 10     | TEBAGE | 1300.0   | 5    |
| 10     | MILLER | 1300.0   | 5    |
| 20     | SCOTT  | 3000.0   | 1    |
| 20     | FORD   | 3000.0   | 1    |
| 20     | JONES  | 2975.0   | 3    |
| 20     | ADAMS  | 1100.0   | 4    |
| 20     | SMITH  | 800.0    | 5    |
| 30     | BLAKE  | 2850.0   | 1    |
| 30     | ALLEN  | 1600.0   | 2    |
| 30     | TURNER | 1500.0   | 3    |
| 30     | MARTIN | 1250.0   | 4    |
| 30     | WARD   | 1250.0   | 4    |
| 30     | JAMES  | 950.0    | 6    |
+-----+-----+-----+-----+

```

6.6.4.12. LAG

This topic describes the LAG function and provides examples of using this function.

Syntax:

```
lag(expr, Bigint offset, default) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]])
```

Description: This function returns the values for a row at a given offset before the current row. For example, if the current row is m , the values for the $m - \text{offset}$ th row are returned.

Parameters:

- `expr`: a value of any data type.
- `offset`: the offset, which is a constant of the BIGINT type. The value of the offset must be greater than 0. If the input value is of the STRING or DOUBLE type, it is implicitly converted into the BIGINT type before calculation.
- `default`: The default value when the offset is out of the valid range. The value of this parameter must be a constant. The default value of this parameter is NULL.
- `partition by [col1, col2...]`: the columns in the window that is used for calculation.
- `order by col1[asc|desc], col2[asc|desc]`: specifies how values in the rows are sorted.

Return value: A value of the same data type as `expr` is returned.

Examples:

```
select seq, lag(seq+100, 1) over (partition by window order by seq) as r from sliding_window;
-- The following result is returned:
+-----+-----+
| seq   | r     |
+-----+-----+
| 0     | NULL  |
| 1     | 100   |
| 2     | 101   |
| 3     | 102   |
| 4     | 103   |
| 5     | 104   |
| 6     | 105   |
| 7     | 106   |
| 8     | 107   |
| 9     | 108   |
+-----+-----+
```

6.6.4.13. LEAD

This topic describes the LEAD function and provides examples of using this function.

Syntax:

```
lead(expr, Bigint offset, default) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]])
```

Description: This function returns the values for a row at a given offset after the current row. For example, if the current row is m , the values for the $m + \text{offset}$ th row are returned.

Parameters:

- `expr`: a value of any data type.
- `offset`: the offset, which is a constant of the BIGINT type. The value of this parameter must be greater than 0. If the input value is of the STRING or DOUBLE type, it is implicitly converted into the BIGINT type.
- `default`: the default value when the value of `offset` is out of the valid range. The value of this parameter is a constant. The default value of this parameter is NULL.

- `partition by [col1, col2...]`: the columns in the window that is used for calculation.
- `order by [col1[asc|desc], col2[asc|desc]]`: specifies how return values are sorted.

Return value: A value of the same data type as `expr` is returned.

Examples:

Column information in the `test_lead` table:

`c_int_a,c_Double_a,c_String_a,c_String_b,c_time_a,c_time_b,c_String_in_fact_num`.

```
select c_double_a,c_string_b,c_int_a,lead(c_int_a,1) over(partition by c_double_a order by c_string_b) from test_lead;
select c_string_a,c_time_b,c_double_a,lead(c_double_a,1) over(partition by c_string_a order by c_time_b) from test_lead;
select c_string_in_fact_num,c_string_a,c_int_a,lead(c_int_a) over(partition by c_string_in_fact_num order by c_string_a) from test_lead;
```

6.6.4.14. PERCENT_RANK

This topic describes the `PERCENT_RANK` function.

Syntax:

```
percent_rank() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])
```

Description: This function returns the relative percent rank of a row in a group of data.

Parameters:

- `partition by [col1, col2...]`: the columns in the window that is used for calculation.
- `order by col1[asc|desc], col2[asc|desc]`: the value that is used to calculate the percent rank.

Return value: A value of the `DOUBLE` type is returned. Valid values: `[0,1]`. The relative percent rank is calculated by using the following formula: $(\text{Rank} - 1) / (\text{Number of rows} - 1)$

 **Note** The number of rows in a single window cannot exceed 10 million.

6.6.4.15. ROW_NUMBER

This topic describes the `ROW_NUMBER` function and provides examples of using this function.

Syntax:

```
row_number() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])
```

Description: This function returns the ordinal number of the current row within a group of rows, counting from 1.

Parameters:

- `partition by [col1, col2...]`: the columns in the window that is used for calculation.
- `order by [col1[asc|desc], col2[asc|desc]]`: the values that need to be sorted to return the ordinal number of the current row.

Return value: A value of the `BIGINT` type is returned.

Examples:

The `emp` table contains the following data:

```

| empno | ename | job | mgr | hiredate | sal | comm | deptno |
7369, SMITH, CLERK, 7902, 1980-12-17 00:00:00, 800, , 20
7499, ALLEN, SALESMAN, 7698, 1981-02-20 00:00:00, 1600, 300, 30
7521, WARD, SALESMAN, 7698, 1981-02-22 00:00:00, 1250, 500, 30
7566, JONES, MANAGER, 7839, 1981-04-02 00:00:00, 2975, , 20
7654, MARTIN, SALESMAN, 7698, 1981-09-28 00:00:00, 1250, 1400, 30
7698, BLAKE, MANAGER, 7839, 1981-05-01 00:00:00, 2850, , 30
7782, CLARK, MANAGER, 7839, 1981-06-09 00:00:00, 2450, , 10
7788, SCOTT, ANALYST, 7566, 1987-04-19 00:00:00, 3000, , 20
7839, KING, PRESIDENT, , 1981-11-17 00:00:00, 5000, , 10
7844, TURNER, SALESMAN, 7698, 1981-09-08 00:00:00, 1500, 0, 30
7876, ADAMS, CLERK, 7788, 1987-05-23 00:00:00, 1100, , 20
7900, JAMES, CLERK, 7698, 1981-12-03 00:00:00, 950, , 30
7902, FORD, ANALYST, 7566, 1981-12-03 00:00:00, 3000, , 20
7934, MILLER, CLERK, 7782, 1982-01-23 00:00:00, 1300, , 10
7948, JACCKA, CLERK, 7782, 1981-04-12 00:00:00, 5000, , 10
7956, WELAN, CLERK, 7649, 1982-07-20 00:00:00, 2450, , 10
7956, TEBAGE, CLERK, 7748, 1982-12-30 00:00:00, 1300, , 10

```

Group employees by department, sort the employees in each group in descending order of sal, and then obtain the ordinal numbers of employees in each group.

```

select deptno
       , ename
       , sal
       , row_number() over (partition by deptno order by sal desc) as nums
-- DEPTNO (department) is the column in the window that is used for calculation. Values in the sal c
column are sorted to return the ordinal number of the current row.
from emp;
-- The following result is returned:
+-----+-----+-----+-----+
| deptno | ename | sal      | nums |
+-----+-----+-----+-----+
| 10     | JACCKA | 5000.0   | 1    |
| 10     | KING   | 5000.0   | 2    |
| 10     | CLARK  | 2450.0   | 3    |
| 10     | WELAN  | 2450.0   | 4    |
| 10     | TEBAGE | 1300.0   | 5    |
| 10     | MILLER | 1300.0   | 6    |
| 20     | SCOTT  | 3000.0   | 1    |
| 20     | FORD   | 3000.0   | 2    |
| 20     | JONES  | 2975.0   | 3    |
| 20     | ADAMS  | 1100.0   | 4    |
| 20     | SMITH  | 800.0    | 5    |
| 30     | BLAKE  | 2850.0   | 1    |
| 30     | ALLEN  | 1600.0   | 2    |
| 30     | TURNER | 1500.0   | 3    |
| 30     | MARTIN | 1250.0   | 4    |
| 30     | WARD   | 1250.0   | 5    |
| 30     | JAMES  | 950.0    | 6    |
+-----+-----+-----+-----+

```

6.6.4.16. CLUSTER_SAMPLE

This topic describes the CLUSTER_SAMPLE function and provides examples of using this function.

Syntax:

```
boolean cluster_sample([Bigint x, Bigint y])
over(partition by [col1, col2..])
```

Description: This function performs cluster sampling.

Parameters:

- **x**: a constant of the BIGINT type. The value of this parameter must be greater than or equal to 1. If y is specified, x indicates that a window is divided into x portions. Otherwise, x indicates that the records of x rows in a window are extracted. In this case, True is returned for the x rows. If the value of x is NULL, NULL is returned.
- **y**: a constant of the BIGINT type. The value of y must be greater than or equal to 1 and less than or equal to x. y indicates that y records of the x portions in a window are extracted. In this case, True is returned for the y records. If the value of y is NULL, NULL is returned.
- **partition by [col1, col2..]**: the columns in the window that is used for calculation.

Return value: A value of the BOOLEAN type is returned.

Examples:

The test_tbl table contains two columns: key and value. key specifies a group, which can be groupa or groupb. value indicates the value of key. Data in the test_tbl table:

key	value
groupa	-1.34764165478145
groupa	0.740212609046718
groupa	0.167537127858695
groupa	0.630314566185241
groupa	0.0112401388646925
groupa	0.199165745875297
groupa	-0.320543343353587
groupa	-0.273930924365012
groupa	0.386177958942063
groupa	-1.09209976687047
groupb	-1.10847690938643
groupb	-0.725703978381499
groupb	1.05064697475759
groupb	0.135751224393789
groupb	2.13313102040396
groupb	-1.11828960785008
groupb	-0.849235511508911
groupb	1.27913806620453
groupb	-0.330817716670401
groupb	-0.300156896191195
groupb	2.4704244205196
groupb	-1.28051882084434

If you want to extract a sample of 10% of the values from each group, execute the following MaxCompute SQL statement:

```

select key, value from (select key, value, cluster_sample(10, 1) over(partition by key) as flag from
tbl) sub where flag = true;
-- The following result is returned:
+-----+-----+
| key   | value           |
+-----+-----+
| groupa | -0.273930924365012 |
| groupb | -1.11828960785008 |
+-----+-----+

```

6.6.4.17. NTILE

This topic describes the NTILE function and provides examples of using this function.

Syntax:

```

BIGINT ntile(BIGINT n) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])

```

Description: This function divides rows into n ranking groups of as equal size as possible and returns the ranking group that a given row falls into. If rows are not evenly divided into ranking groups, more rows are included in the first ranking group.

Parameters: n: a value of the BIGINT type.

Return value: A value of the BIGINT type is returned.

Examples:

The emp table contains the following data:

```

| empno | ename | job | mgr | hiredate | sal | comm | deptno |
7369, SMITH, CLERK, 7902, 1980-12-17 00:00:00, 800, , 20
7499, ALLEN, SALESMAN, 7698, 1981-02-20 00:00:00, 1600, 300, 30
7521, WARD, SALESMAN, 7698, 1981-02-22 00:00:00, 1250, 500, 30
7566, JONES, MANAGER, 7839, 1981-04-02 00:00:00, 2975, , 20
7654, MARTIN, SALESMAN, 7698, 1981-09-28 00:00:00, 1250, 1400, 30
7698, BLAKE, MANAGER, 7839, 1981-05-01 00:00:00, 2850, , 30
7782, CLARK, MANAGER, 7839, 1981-06-09 00:00:00, 2450, , 10
7788, SCOTT, ANALYST, 7566, 1987-04-19 00:00:00, 3000, , 20
7839, KING, PRESIDENT, , 1981-11-17 00:00:00, 5000, , 10
7844, TURNER, SALESMAN, 7698, 1981-09-08 00:00:00, 1500, 0, 30
7876, ADAMS, CLERK, 7788, 1987-05-23 00:00:00, 1100, , 20
7900, JAMES, CLERK, 7698, 1981-12-03 00:00:00, 950, , 30
7902, FORD, ANALYST, 7566, 1981-12-03 00:00:00, 3000, , 20
7934, MILLER, CLERK, 7782, 1982-01-23 00:00:00, 1300, , 10
7948, JACCKA, CLERK, 7782, 1981-04-12 00:00:00, 5000, , 10
7956, WELAN, CLERK, 7649, 1982-07-20 00:00:00, 2450, , 10
7956, TEBAGE, CLERK, 7748, 1982-12-30 00:00:00, 1300, , 10

```

Classify all employees into three groups by department, sort employees in each group in descending order of sal, and then obtain the ranking groups of employees in each group.

```
select deptno,ename,sal,ntile(3) over(partition by deptno order by sal desc) as nt3 from emp;
-- The following result is returned:
+-----+-----+-----+-----+
| deptno | ename | sal      | nt3      |
+-----+-----+-----+-----+
| 10     | JACCKA | 5000.0   | 1        |
| 10     | KING   | 5000.0   | 1        |
| 10     | WELAN  | 2450.0   | 2        |
| 10     | CLARK  | 2450.0   | 2        |
| 10     | TEBAGE | 1300.0   | 3        |
| 10     | MILLER | 1300.0   | 3        |
| 20     | SCOTT  | 3000.0   | 1        |
| 20     | FORD   | 3000.0   | 1        |
| 20     | JONES  | 2975.0   | 2        |
| 20     | ADAMS  | 1100.0   | 2        |
| 20     | SMITH  | 800.0    | 3        |
| 30     | BLAKE  | 2850.0   | 1        |
| 30     | ALLEN  | 1600.0   | 1        |
| 30     | TURNER | 1500.0   | 2        |
| 30     | MARTIN | 1250.0   | 2        |
| 30     | WARD   | 1250.0   | 3        |
| 30     | JAMES  | 950.0    | 3        |
+-----+-----+-----+-----+
```

6.6.4.18. NTH_VALUE

This topic describes the NTH_VALUE function and provides examples of using this function.

Syntax:

```
nth_value(expr, number [, skipNull]) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

Description: This function calculates the nth value in the window. If the value of n exceeds the total number of rows in the window, NULL is returned.

Parameters:

- **expr**: a value of any basic data type.
- **number**: an integer that is greater than or equal to 1.
- **skipNull**: specifies whether to ignore the rows with NULL values when you calculate the nth value. The value of this parameter is of the BOOLEAN type. The default value is False.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]]**: If ORDER BY is not specified, the value of expr of the nth row in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the value of expr of the nth row from the starting row to the current row in the current window is returned.

Return value: A value of the same data type as expr is returned.

Examples:

```
select user_id, price, nth_value(price, 2) over
(partition by user_id) as nth_value from test_src;
+-----+-----+-----+
| user_id | price | nth_value |
+-----+-----+-----+
| 1       | 5.5   | 4.5       |
| 1       | 4.5   | 4.5       | -- This row is the second row in the current window.
```

```

| 1      | 6.5      | 4.5      |
| 1      | 5.5      | 4.5      |
| 2      | NULL     | 3.0      |
| 2      | 3.0      | 3.0      | -- This row is the second row in the current window.
| 3      | 4.0      | NULL     |
| 3      | NULL     | NULL     | -- This row is the second row in the current window.
+-----+-----+-----+
-- If ORDER BY is not specified, rows from the first row to the last row are in the current window.
The value of the second row is returned.
select user_id, price, nth_value(price, 3) over
  (partition by user_id) as nth_value from test_src;
+-----+-----+-----+
| user_id | price  | nth_value |
+-----+-----+-----+
| 1      | 5.5    | 6.5      |
| 1      | 4.5    | 6.5      |
| 1      | 6.5    | 6.5      | -- This row is the third row in the current window.
| 1      | 5.5    | 6.5      |
| 2      | NULL   | NULL     |
| 2      | 3.0    | NULL     |
| 3      | 4.0    | NULL     |
| 3      | NULL   | NULL     |
+-----+-----+-----+
-- If ORDER BY is not specified, rows from the first row to the last row are in the current window.
The value of the third row is returned.
-- The second and third windows have only two rows.
select user_id, price, nth_value(price, 2) over
  (partition by user_id order by price) as nth_value from test_src;
+-----+-----+-----+
| user_id | price  | nth_value |
+-----+-----+-----+
| 1      | 4.5    | NULL     | -- The current window has only one row. The second row exceeds
the window length.
| 1      | 5.5    | 5.5      |
| 1      | 5.5    | 5.5      |
| 1      | 6.5    | 5.5      |
| 2      | NULL   | NULL     |
| 2      | 3.0    | 3.0      |
| 3      | NULL   | NULL     |
| 3      | 4.0    | 4.0      |
+-----+-----+-----+
-- If ORDER BY is specified, rows from the first row to the current row belong to the current window
. The value of the second row is returned.
select user_id, price, nth_value(price, 1, true) over
  (partition by user_id) as nth_value from test_src;
+-----+-----+-----+
| user_id | price  | nth_value |
+-----+-----+-----+
| 1      | 5.5    | 5.5      |
| 1      | 4.5    | 5.5      |
| 1      | 6.5    | 5.5      |
| 1      | 5.5    | 5.5      |
| 2      | NULL   | 3.0      | -- The value of the first row is NULL, and therefore this row
is skipped.
| 2      | 3.0    | 3.0      |
| 3      | 4.0    | 4.0      |
| 3      | NULL   | 4.0      |
+-----+-----+-----+
-- If ORDER BY is not specified, rows from the first row to the last row belong to the current window

```

```
w.
-- The value of the first row is returned. skipNull is set to True.
```

6.6.4.19. CUME_DIST

This topic describes the CUME_DIST function and provides examples of using this function.

Syntax:

```
cume_dist() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...]
```

Description: This function calculates the cumulative distribution. The cumulative distribution is the ratio of rows whose values are greater than or equal to the current value to all rows in a group.

Parameters: ORDER BY: the value used for comparison.

Return value: The ratio of the number of rows whose values are greater than or equal to the current value in the group to the total number of rows in the group is returned.

Examples:

The emp table contains the following data:

```
| empno | ename | job | mgr | hiredate | sal | comm | deptno |
7369, SMITH, CLERK, 7902, 1980-12-17 00:00:00, 800, , 20
7499, ALLEN, SALESMAN, 7698, 1981-02-20 00:00:00, 1600, 300, 30
7521, WARD, SALESMAN, 7698, 1981-02-22 00:00:00, 1250, 500, 30
7566, JONES, MANAGER, 7839, 1981-04-02 00:00:00, 2975, , 20
7654, MARTIN, SALESMAN, 7698, 1981-09-28 00:00:00, 1250, 1400, 30
7698, BLAKE, MANAGER, 7839, 1981-05-01 00:00:00, 2850, , 30
7782, CLARK, MANAGER, 7839, 1981-06-09 00:00:00, 2450, , 10
7788, SCOTT, ANALYST, 7566, 1987-04-19 00:00:00, 3000, , 20
7839, KING, PRESIDENT, , 1981-11-17 00:00:00, 5000, , 10
7844, TURNER, SALESMAN, 7698, 1981-09-08 00:00:00, 1500, 0, 30
7876, ADAMS, CLERK, 7788, 1987-05-23 00:00:00, 1100, , 20
7900, JAMES, CLERK, 7698, 1981-12-03 00:00:00, 950, , 30
7902, FORD, ANALYST, 7566, 1981-12-03 00:00:00, 3000, , 20
7934, MILLER, CLERK, 7782, 1982-01-23 00:00:00, 1300, , 10
7948, JACCKA, CLERK, 7782, 1981-04-12 00:00:00, 5000, , 10
7956, WELAN, CLERK, 7649, 1982-07-20 00:00:00, 2450, , 10
7956, TEBAGE, CLERK, 7748, 1982-12-30 00:00:00, 1300, , 10
```

Group all employees by department and obtain the cumulative distribution of sal for each group.

```

select deptno
,   ename
,   sal
,   concat(round(cume_dist() over(partition by deptno order by sal desc)*100,2),'%') as cume_dist
from emp;
-- The following result is returned:
+-----+-----+-----+-----+
| deptno | ename | sal      | cume_dist |
+-----+-----+-----+-----+
| 10     | JACCKA | 5000.0   | 33.33%    |
| 10     | KING  | 5000.0   | 33.33%    |
| 10     | CLARK  | 2450.0   | 66.67%    |
| 10     | WELAN  | 2450.0   | 66.67%    |
| 10     | TEBAGE | 1300.0   | 100.0%    |
| 10     | MILLER | 1300.0   | 100.0%    |
| 20     | SCOTT  | 3000.0   | 40.0%     |
| 20     | FORD   | 3000.0   | 40.0%     |
| 20     | JONES  | 2975.0   | 60.0%     |
| 20     | ADAMS  | 1100.0   | 80.0%     |
| 20     | SMITH  | 800.0    | 100.0%    |
| 30     | BLAKE  | 2850.0   | 16.67%    |
| 30     | ALLEN  | 1600.0   | 33.33%    |
| 30     | TURNER | 1500.0   | 50.0%     |
| 30     | MARTIN | 1250.0   | 83.33%    |
| 30     | WARD   | 1250.0   | 83.33%    |
| 30     | JAMES  | 950.0    | 100.0%    |
+-----+-----+-----+-----+

```

6.6.4.20. FIRST_VALUE

This topic describes the FIRST_VALUE function and provides examples of using this function.

Syntax:

```

first_value(expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause]

```

Description: This function sorts rows and returns the first value in the range from the starting row to the current row.

Parameters:

- **expr**: a value of any basic data type.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]**: If ORDER BY is not specified, the value of expr of the starting row in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the value of expr of the starting row in the current window is returned.

Return value: A value of the same data type as expr is returned.

Examples:

```

select user_id, price, first_value(price) over
  (partition by user_id) as first_value from test_src;
+-----+-----+-----+
| user_id | price  | first_value |
+-----+-----+-----+
| 1       | 5.5    | 5.5         | -- This row is the starting row of this window.
| 1       | 4.5    | 5.5         |
| 1       | 6.5    | 5.5         |
| 1       | 5.5    | 5.5         |
| 2       | NULL   | NULL        | -- This row is the starting row of this window.
| 2       | 3.0    | NULL        |
| 3       | 4.0    | 4.0         | -- This row is the starting row of this window.
| 3       | NULL   | 4.0         |
+-----+-----+-----+
-- If ORDER BY is not specified, rows from the first row to the last row belong to the current window.
The value of the starting row in the current window is returned.
select user_id, price, first_value(price) over
  (partition by user_id order by price) as first_value from test_src;
+-----+-----+-----+
| user_id | price  | first_value |
+-----+-----+-----+
| 1       | 4.5    | 4.5         | -- This row is the starting row of this window.
| 1       | 5.5    | 4.5         |
| 1       | 5.5    | 4.5         |
| 1       | 6.5    | 4.5         |
| 2       | NULL   | NULL        | -- This row is the starting row of this window.
| 2       | 3.0    | NULL        |
| 3       | NULL   | NULL        | -- This row is the starting row of this window.
| 3       | 4.0    | NULL        |
+-----+-----+-----+
-- If ORDER BY is specified, rows from the first row to the current row belong to the current window.
. The value of the starting row in the current window is returned.

```

6.6.4.21. LAST_VALUE

This topic describes the LAST_VALUE function and provides examples of using this function.

Syntax:

```

last_value(expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])

```

Description: This function sorts rows and returns the last value in the range from the starting row to the current row.

Parameters:

- **expr**: a value of any basic data type.
- **partition by [col1, col2...]**: the columns in the window that is used for calculation.
- **order by [col1[asc|desc], col2[asc|desc]**: If ORDER BY is not specified, the value of expr of the last row in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the value of expr of the current row in the current window is returned.

Return value: A value of the same data type as expr is returned.

Examples:

```

select user_id, price, last_value(price) over
  (partition by user_id) as last_value from test_src;
+-----+-----+-----+
| user_id | price | last_value |
+-----+-----+-----+
| 1       | 5.5   | 5.5       |
| 1       | 4.5   | 5.5       |
| 1       | 6.5   | 5.5       |
| 1       | 5.5   | 5.5       | -- This row is the last row in this window.
| 2       | NULL  | 3.0       |
| 2       | 3.0   | 3.0       | -- This row is the last row in this window.
| 3       | 4.0   | NULL      |
| 3       | NULL  | NULL      | -- This row is the last row in this window.
+-----+-----+-----+
-- If ORDER BY is not specified, the rows from the first row to the last row belong to the current w
indow, and the value of the last row in the current window is returned.
select user_id, price, last_value(price) over
  (partition by user_id order by price) as last_value from test_src;
+-----+-----+-----+
| user_id | price | last_value |
+-----+-----+-----+
| 1       | 4.5   | 4.5       | -- This row is the current row in the current window.
| 1       | 5.5   | 5.5       | -- This row is the current row in the current window.
| 1       | 5.5   | 5.5       | -- This row is the current row in the current window.
| 1       | 6.5   | 6.5       | -- This row is the current row in the current window.
| 2       | NULL  | NULL      | -- This row is the current row in the current window.
| 2       | 3.0   | 3.0       | -- This row is the current row in the current window.
| 3       | NULL  | NULL      | -- This row is the current row in the current window.
| 3       | 4.0   | 4.0       | -- This row is the current row in the current window.
+-----+-----+-----+
-- If ORDER BY is specified, rows from the first row to the current row belong to the current window
. The value of the current row in the current window is returned.

```

6.6.5. Aggregate functions

6.6.5.1. Expressions of filter conditions in aggregate functions

This topic describes the expressions of filter conditions in aggregate functions and provides examples.

Filter conditions can be added for all aggregate functions. If FILTER is specified, only the rows for which the value of where_condition is TRUE are processed by the required aggregate functions.

Syntax:

```
aggregate_name(expression[,...]) [FILTER (WHERE where_condition)]
```

Examples:

```

SELECT
  SUM(x),
  SUM(x) FILTER (WHERE y > 1),
  SUM(x) FILTER (WHERE y > 2)
FROM VALUES (NULL, 1), (1,2), (2,3), (3,NULL) AS t(x,y);

```

The following result is returned:

```

+-----+-----+-----+
| _c0   | _c1   | _c2   |
+-----+-----+-----+
| 6     | 3     | 2     |
+-----+-----+-----+
    
```

Note

- Only built-in aggregate functions support FILTER (WHERE where_condition). User-defined aggregate functions (UDAFs) do not support FILTER (WHERE where_condition).
- COUNT(*) cannot be used with FILTER (WHERE where_condition). To implement this feature, use the COUNT_IF function.

6.6.5.2. COUNT

This topic describes the COUNT function and provides examples of using this function.

Syntax:

```
bigint count([distinct|all] value)
```

Description: This function returns the number of records.

Parameters:

- distinct|all: specifies whether to deduplicate records during counting. The default value is all, which indicates that all records are counted. If this parameter is set to distinct, only records with distinct values are counted.
- value: a value of any data type. If the input value for a row is NULL, this row is not used for calculation. You can set value to an asterisk (*). This means that you can run count(*) to count all records.

Return value: A value of the BIGINT type is returned.

Examples:

The tbla table contains the col1 column of the BIGINT type. Data in the tbla table:

```

+-----+
| col1 |
+-----+
| 1    |
+-----+
| 2    |
+-----+
| NULL |
+-----+
    
```

Execute the following SQL statements:

```

select count(*) from tbla;
-- 3 is returned.
select count(col1) from tbla;
-- 2 is returned.
    
```

The COUNT function can be used with the GROUP BY clause. For example, the test_src table contains two columns: key and value. key is of the STRING type and value is of the DOUBLE type.

Data in the test_src table:

```

+-----+-----+
| key | value |
+-----+-----+
| a   | 2.0   |
+-----+-----+
| a   | 4.0   |
+-----+-----+
| b   | 1.0   |
+-----+-----+
| b   | 3.0   |
+-----+-----+

```

Execute the following statement:

```

select key, count(value) as count from test_src group by key;
-- The following result is returned:
+-----+-----+
| key | count |
+-----+-----+
| a   | 2     |
+-----+-----+
| b   | 2     |
+-----+-----+

```

The COUNT function aggregates the values with the same key. The usage of other aggregate functions is the same as that of this function and is not described in detail in this document.

6.6.5.3. COUNT_IF

This topic describes the aggregate function COUNT_IF and provides examples.

Syntax:

```
bigint count_if(BOOLEAN expr)
```

Description: This function calculates the number of records whose value of expr is TRUE.

Parameters: expr, which is of the BOOLEAN type. If the value for a row is FALSE or NULL, the row is not counted.

Return value: A value of the BIGINT type is returned.

Examples:

```
SELECT COUNT_IF(x > 1), COUNT_IF(x <=1) FROM VALUES (NULL), (0), (1), (2) AS t(x);
```

The following result is returned:

```

+-----+-----+
| _c0 | _c1 |
+-----+-----+
| 1   | 2   |
+-----+-----+

```

6.6.5.4. AVG

This topic describes the AVG function and provides examples of using this function.

Syntax:

```
double avg(double value)
decimal avg(decimal value)
```

Description: This function returns the average value of a column.

Parameters: value: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If the input value is not of the STRING or BIGINT type, an error is returned. If the input value for a row is NULL, this row is not used for calculation. Values of the BOOLEAN type cannot be used for calculation.

Return value: If the input value is of the DECIMAL type, a value of the DECIMAL type is returned. If the input value is not of the DECIMAL type, a value of the DOUBLE type is returned.

Examples:

The tbla table contains the value column of the BIGINT type. Data in the tbla table:

```
+-----+
| value |
+-----+
| 1     |
| 2     |
| NULL  |
+-----+
```

Execute the following statement:

```
select avg(value) as avg from tbla;
-- The average value of the value column is 1.5, which is calculated by using the following formula:
(1 + 2)/2
+-----+
| avg  |
+-----+
| 1.5  |
+-----+
```

6.6.5.5. MAX

This topic describes the MAX function and provides examples of using this function.

Syntax:

```
max(value)
```

Description: This function calculates the maximum value of values in a column.

Parameters: value: a value of any data type. If the value for a row is NULL, this row that corresponds to the column is not used for calculation. Values of the BOOLEAN type are not used for calculation.

Return value: A value of the same data type as value is returned.

Examples:

The tbla table contains the col1 column of the BIGINT type. Data in the tbla table:

```
+-----+
| coll |
+-----+
| 1    |
+-----+
| 2    |
+-----+
| NULL |
+-----+
```

Execute the following statement:

```
select max(value) from tbla;
-- 2 is returned.
```

6.6.5.6. MIN

This topic describes the MIN function and provides examples of using this function.

Syntax:

```
min(value)
```

Description: This function calculates the minimum value of values in a column.

Parameters: value: a value of any data type. If the value for a row is NULL, this row that corresponds to the column is not used for calculation. Values of the BOOLEAN type are not used for calculation.

Return value: A value of the same data type as value is returned.

Examples:

The tbla table contains the value column of the BIGINT type. Data in the tbla table:

```
+-----+
| value|
+-----+
| 1    |
+-----+
| 2    |
+-----+
| NULL |
+-----+
```

Execute the following statement:

```
select min(value) from tbla;
-- 1 is returned.
```

6.6.5.7. MEDIAN

This topic describes the MEDIAN function and provides examples of using this function.

Syntax:

```
double median(double number)
decimal median(decimal number)
```

Description: This function calculates the median value for a group of values.

Parameters: value: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If the input value is not of the STRING or BIGINT type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned.

Examples:

The tbla table contains the value column of the BIGINT type. Data in the tbla table:

```
+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| 3 |
+-----+
| 4 |
+-----+
| 5 |
+-----+
```

Execute the following statement:

```
select median(value) from tbla;
-- 3.0 is returned.
```

6.6.5.8. STDDEV

This topic describes the STDDEV function and provides examples of using this function.

Syntax:

```
double stddev(double number)
decimal stddev(decimal number)
```

Description: This function calculates the population standard deviation.

Parameters: value: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If the input value is not of the STRING or BIGINT type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned.

Examples:

The tbla table contains the value column of the BIGINT type. Data in the tbla table:

```
+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| 3 |
+-----+
| 4 |
+-----+
| 5 |
+-----+
```

Execute the following statement:

```
select stddev(value) from tbla;
-- 1.4142135623730951 is returned.
```

6.6.5.9. STDDEV_SAMP

This topic describes the STDDEV_SAMP function and provides examples of using this function.

Syntax:

```
double stddev_samp(double number)
decimal stddev_samp(decimal number)
```

Description: This function returns the sample standard deviation of a group of values.

Parameters: value: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If the input value is not of the STRING or BIGINT type, an error is returned.

Return value: A value of the DOUBLE or DECIMAL type is returned.

Examples:

The tbla table contains the value column of the BIGINT type. Data in the tbla table:

```
+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| 3 |
+-----+
| 4 |
+-----+
| 5 |
+-----+
```

Execute the following statement:

```
select stddev_samp(value) from tbla;
-- 1.58113883008418981 is returned.
```

6.6.5.10. SUM

This topic describes the SUM function and provides examples of using this function.

Syntax:

```
sum(value)
```

Description: This function returns the sum of values in specified rows.

Parameters: value: a value of the DOUBLE, DECIMAL, or BIGINT type. If the input value is of the STRING type, it is implicitly converted into the DOUBLE type before calculation. If the input value for a row is NULL, this row is not used for calculation. Values of the BOOLEAN type are not used for calculation.

Return value: If the input value is of the BIGINT type, a value of the BIGINT type is returned. If the input value is of the DOUBLE or STRING type, a value of the DOUBLE type is returned.

Examples:

The tbla table contains the value column of the BIGINT type. Data in the tbla table:

```
+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| NULL |
+-----+
```

Execute the following statement:

```
select sum(value) from tbla;
-- 3 is returned.
```

6.6.5.11. WM_CONCAT

This topic describes the WM_CONCAT function and provides examples of using this function.

Syntax:

```
string wm_concat(string separator, string str)
```

Description: This function returns a string of values that are separated by using a given delimiter. The delimiter is specified by separator.

Parameters:

- separator: the delimiter, which is a constant of the STRING type. If it is not of the STRING type or is not a constant, an error is returned.
- str: a value of the STRING type. If the input value is of the BIGINT, DOUBLE, or DATETIME type, it is implicitly converted into the STRING type before calculation. If the input value is not of the BIGINT, DOUBLE, or DATETIME type, an error is returned.

Return value: A value of the STRING type is returned.

 **Note** If test_src in select wm_concat(',', name) from test_src; is an empty set, NULL is returned.

Examples:

Group and sort values in the test table and use a string to list the values in the same group.

```
-- Create the test table.
create table test(id int , alphabet string);
-- Insert data into the test table.
insert into test values (1,'a'), (1,'b'), (1,'c'), (2,'D'), (2,'E'), (2,'F');
-- Group and sort values in the test table based on the id column. Then, use a string to list values
in the same group.
select id,wm_concat(',',alphabet) from test group by id order by id limit 100;
+-----+-----+
| id      | _c1      |
+-----+-----+
| 1       | abc      |
| 2       | DEF      |
+-----+-----+
```

6.6.5.12. PERCENTILE

This topic describes the PERCENTILE function and provides examples of using this function.

Syntax:

```
double percentile(bigint col, p)
array<double> percentile(bigint col, array(p1 [, p2]...))
```

Description: This function returns the pth percentile of the specified column. p must be between 0 and 1.

 **Notice** You can calculate the percentile only for integer values.

Parameters:

- col: the name of the table column of the BIGINT type.
- p: the percentile that must be between 0 and 1.

Examples:

The var_test table contains the c1 column. Data in the var_test table:

```
+-----+
| c1      |
+-----+
| 8       |
| 9       |
| 10      |
| 11      |
+-----+
```

Execute the following statement to calculate the pth percentile of the c1 column.

```

select percentile(c1,0),percentile(c1,0.3),percentile(c1,0.5),percentile(c1,1) from var_test;
-- The following result is returned:
+-----+-----+-----+-----+
| _c0 | _c1 | _c2 | _c3 |
+-----+-----+-----+-----+
| 8.0 | 8.9 | 9.5 | 11.0 |
+-----+-----+-----+-----+
select percentile(c1,array(0,0.3,0.5,1))from var_test;
-- The following result is returned:
+-----+
| _c0 |
+-----+
| [8, 8.9, 9.5, 11] |
+-----+

```

6.6.5.13. Additional aggregate functions

6.6.5.13.1. Usage notes

This topic describes the configuration operation that you must perform before you use additional aggregate functions.

MaxCompute V2.0 provides more aggregate functions. You must add the following SET statement before the SQL statements for these new functions:

```
set odps.sql.type.system.odps2=true;
```

 **Note** You must submit and execute the SET statement and the SQL statements for the new functions at the same time.

6.6.5.13.2. COLLECT_LIST

This topic describes the COLLECT_LIST function and provides examples of using this function.

Syntax:

```
ARRAY collect_list(col)
```

Description: This function aggregates the expressions that are specified by col in a given group into an array.

Parameters: col: a column of the table, which is of any data type.

Return value: A value of the ARRAY type is returned.

6.6.5.13.3. COLLECT_SET

This topic describes the COLLECT_SET function.

Syntax:

```
array collect_set(col)
```

Description: This function aggregates the expressions that are specified by col in a given group into an array without duplicate elements.

Parameters: col: a table column, which can be of any data type.

Return value: A value of the ARRAY type is returned.

6.6.5.13.4. VARIANCE or VAR_POP

This topic describes the VARIANCE or VAR_POP function and provides examples of using this function.

Syntax:

```
DOUBLE variance(col)
DOUBLE var_pop(col)
```

Description: This function calculates the variance of a specified column of a numeric data type.

Parameters: col: the column of a numeric data type. If it is not of a numeric data type, NULL is returned.

Return value: A value of the DOUBLE type is returned.

Examples:

The test table contains the c1 column. Data in the test table:

```
+-----+
| c1    |
+-----+
| 8     |
| 9     |
| 10    |
| 11    |
+-----+
```

Execute the following statement to calculate the variance of the c1 column:

```
select variance(c1) from test;
-- You can also execute the following statement:
select var_pop(c1) from test;
-- The following result is returned:
+-----+
| _c0 |
+-----+
| 1.25 |
+-----+
```

6.6.5.13.5. VAR_SAMP

This topic describes the VAR_SAMP function and provides examples of using this function.

Syntax:

```
DOUBLE var_samp(col)
```

Description: This function calculates the sample variance of a specified column of a numeric data type.

Parameters: col: a column of a numeric data type. If it is not of a numeric data type, NULL is returned.

Return value: A value of the DOUBLE type is returned

Examples:

The test table contains the c1 column. Data in the test table:

```

+-----+
| c1    |
+-----+
| 8     |
| 9     |
| 10    |
| 11    |
+-----+

```

Execute the following statement to calculate the sample variance of the c1 column:

```

select var_samp(c1) from test;
-- The following result is returned:
+-----+
| _c0 |
+-----+
| 1.6666666666666667 |
+-----+

```

6.6.5.13.6. COVAR_POP

This topic describes the COVAR_POP function and provides examples of using this function.

Syntax:

```
DOUBLE covar_pop(col1, col2)
```

Description: This function calculates the population covariance of two columns of a numeric data type.

Parameters: col1 and col2: the columns of a numeric data type. If they are not of a numeric data type, NULL is returned.

Return value: A value of the DOUBLE type is returned.

Examples:

The test table contains the c1 and c2 columns. Data in the test table:

```

+-----+-----+
| c1    | c2    |
+-----+-----+
| 3     | 2     |
| 14    | 5     |
| 50    | 14    |
| 26    | 75    |
+-----+-----+

```

Execute the following statement to calculate the population covariance of the c1 and c2 columns:

```

select covar_pop(c1,c2) from test;
-- The following result is returned:
+-----+
| _c0 |
+-----+
| 123.49999999999997 |
+-----+

```

6.6.5.13.7. COVAR_SAMP

This topic describes the COVAR_SAMP function and provides examples of using this function.

Syntax:

```
DOUBLE covar_samp(col1, col2)
```

Description: This function calculates the sample covariance of two columns of a numeric data type.

Parameters: col1 and col2: the columns of a numeric data type. If they are not of a numeric data type, NULL is returned.

Return value: A value of the DOUBLE type is returned

Examples:

The test table contains the c1 and c2 columns. Data in the test table:

```
+-----+-----+
| c1      | c2      |
+-----+-----+
| 3       | 2       |
| 14      | 5       |
| 50      | 14      |
| 26      | 75      |
+-----+-----+
```

Execute the following statement to calculate the sample covariance of the c1 and c2 columns:

```
select covar_samp(c1,c2) from test;
-- The following result is returned:
+-----+
| _c0      |
+-----+
| 164.66666666666663 |
+-----+
```

6.6.5.13.8. ANY_VALUE

This topic describes the ANY_VALUE function and provides examples of using this function.

Syntax:

```
ANY_VALUE (value)
```

Description: This function returns a non-deterministic value from a specific column.

Parameters: value: a value of any data type. If the input value for a row is NULL, this row is not used for calculation.

Return value: A value of the same data type as the input value is returned.

Examples:

```
select key, ANY_VALUE(value)
from values (1, 'value1'), (1, 'value2'), (2, 'value3'), (2, NULL), (3, NULL) as t(key, value)
group by key;
-- The following result is returned:
+-----+-----+
| key    | _c1    |
+-----+-----+
| 1      | value1  |
| 2      | value3  |
| 3      | NULL    |
+-----+-----+
```

 **Note** The value returned by ANY_VALUE is not deterministic and different values may be returned for the same input value.

6.6.5.13.9. NUMERIC_HISTOGRAM

This topic describes the NUMERIC_HISTOGRAM function.

Syntax:

```
map<double, double> numeric_histogram(bigint buckets , double value)
```

Description: This function returns the approximate histogram of a given column.

Parameters:

- **buckets**: the maximum number of buckets in the column whose approximate histogram is returned. The value must be of the BIGINT type.
- **value**: the column whose approximate histogram you want to obtain. The column must be of the DOUBLE type.

Return value: A value of the MAP<DOUBLE,DOUBLE> type is returned. In the return value, a key indicates an x-coordinate, and its value indicates the height on the approximate histogram of the column.

6.6.5.13.10. PERCENTILE_APPROX

This topic describes the PERCENTILE_APPROX function and provides examples of using this function.

Syntax:

```
double percentile_approx(double col, p [, B])
array<double> percentile_approx(double col, array(p1 [, p2]...) [, B])
```

Description: This function returns the approximate percentile value of the col column at the given percentage p.

Parameters:

- **p**: the given percentile value. Valid values: [0.0,1.0]. You can specify one percentage to return an approximate percentile value or multiple percentages to return an array that consists of percentile values.
- **B**: the accuracy of the return value. A higher accuracy indicates a more accurate value. If you do not specify this parameter, 10000 is used. If the number of values in the col column is less than B, an exact percentile value is returned.

Return value: percentile_approx(double col, p [, B]) returns a single approximate percentile value.

percentile_approx(double col, array(p1 [, p2]...) [, B]) returns an array that consists of multiple percentile values.

Examples:

```
select percentile_approx(10.0, 0.5, 100);
-- 10.0 is returned.
select percentile_approx(10.0, array(0.5, 0.4, 0.1), 100);
-- [10.0,10.0,10.0] is returned.
```

6.6.5.13.11. APPROX_DISTINCT

This topic describes the APPROX_DISTINCT function and provides examples of using this function.

Syntax:

```
bigint approx_distinct(value)
```

Description: This function returns the approximate number of distinct input values.

Parameters: value: the input data that is used for deduplication.

Return value: A value of the BIGINT type is returned. If all input values are NULL, 0 is returned. This function produces a standard error of 5%.

6.6.6. Other functions

6.6.6.1. ARRAY

This topic describes the ARRAY function and provides examples of using this function.

Syntax:

```
ARRAY array(value1,value2, ...)
```

Description: This function creates an array of a given data type that is specified by value.

Parameters: value: any data type. All the values must be of the same data type.

Return value: A value of the ARRAY type is returned.

Examples:

```
select array(123,456,789);
-- The following result is returned:
[123, 456, 789]
```

6.6.6.2. ARRAY_CONTAINS

This topic describes the ARRAY_CONTAINS function and provides examples of using this function.

Syntax:

```
boolean array_contains(ARRAY<T> a, value v)
```

Description: This function checks whether array a contains value v.

Parameters:

- a: a value of the ARRAY type.
- v: a value of the same data type as other values in the array.

Return value: A value of the BOOLEAN type is returned.

Examples:

```
select array_contains(array('a','b'), 'a');
-- True is returned.
select array_contains(array(456,789),123);
-- False is returned.
```

6.6.6.3. CAST

This topic describes the CAST function.

Syntax:

```
cast(expr as <type>)
```

Description: This function converts the result of an expression into another data type. For example, `cast('1' as bigint)` converts '1' of the STRING type into 1 of the INT type. If the conversion fails, an error is returned.

Notes:

- `cast(double as bigint)` : converts a value of the DOUBLE type to the BIGINT type.
- `cast(string as bigint)` : converts a value of the STRING type to the BIGINT type. If the string consists of numerals expressed as integers, it is directly converted into the BIGINT type. If the string consists of numerals expressed in the FLOAT or EXPONENTIAL form, it is converted into the DOUBLE type and then into the BIGINT type.
- The date format for `cast(string as datetime)` and `cast(datetime as > string)` is yyyy-mm-dd hh:mi:ss.

6.6.6.4. COALESCE

This topic describes the COALESCE function.

Syntax:

```
coalesce(expr1, expr2, ...)
```

Description: This function returns the first non-NULL value in a list. If all values in the list are NULL, NULL is returned.

Parameters: expr: the values you want to test. All these values must be of the same data type or are all NULL. Otherwise, an error is returned.

 **Note** The list must include at least one parameter. Otherwise, an error is returned.

Return value: A value of the same data type as the expr parameter is returned.

6.6.6.5. DECODE

This topic describes the DECODE function and provides examples of using this function.

Syntax:

```
decode(expression, search, result[, search, result]...[, default])
```

Description: This function implements the IF-THEN-ELSE conditional branching feature.

Parameters:

- **expression:** the expression that you want to compare.
- **search:** the search item that you want to compare with expression.
- **result:** the value returned if the values of search and expression match.
- **default:** the value returned if no search item matches the expression. If default is not specified, NULL is returned. This parameter is optional.

Return value: The matched search value is returned. If no matched item exists, default is returned. If default is not specified, NULL is returned.

Note

- Three or more parameters must be specified.
- All results must be of the same data type or be NULL. Inconsistent data types will cause an error. All values of search and expression must be of the same data type. Otherwise, an error is returned.
- If search in the DECODE function has duplicate values that match the expression, the first mapping result is returned.

Examples:

```
select
decode(customer_id,
1, 'Taobao',
2, 'Alipay',
3, 'Aliyun',
NULL, 'N/A',
'Others') as result
FROM VALUES (1,10),
            (1,10),
            (2,null),
            (3,13),
            (4,0),
            (null,6),
            (5,18)
as t(customer_id,value);
```

In this example, the DECODE function implements the IF-THEN-ELSE statement.

```
if customer_id = 1 then
result := 'Taobao';
elsif customer_id = 2 then
result := 'Alipay';
elsif customer_id = 3 then
result := 'Aliyun';
...
else
result := 'Others';
end if;
```

Note

- In most cases, if "NULL = NULL" appears during data computation, the result returned by the MaxCompute SQL engine is NULL, whereas the DECODE function considers that the two NULL values are the same.
- In this example, if the value of customer_id is NULL, the DECODE function returns N/A.

6.6.6.6. EXPLODE

This topic describes the EXPLODE function and provides examples of using this function.

Syntax:

```
explode (var)
```

Description: This function is a user-defined table-valued function (UDTF) that transposes one row into multiple rows.

- If var is of the ARRAY <T> type, the data of the ARRAY type stored in a column is converted into multi-row data.
- If var is of the MAP<K, V> type, each key-value pair of the MAP type stored in a column is transposed to a row with two columns. One column is used for storing keys and the other column is used for storing values.

Parameters: var: a value of the ARRAY<T> or MAP<K, V> type.

Return value: transposed rows.

Note

Limits on the EXPLODE function:

- A SELECT statement can specify only the column that is used for this function. Other columns cannot be specified.
- This function cannot be used with the GROUP BY, CLUSTER BY, DISTRIBUTE BY, or SORT BY clause.

Examples:

```
select explode(array(null, 'a', 'b', 'c'));
-- The following result is returned:
+-----+
| col   |
+-----+
| NULL  |
| a     |
| b     |
| c     |
+-----+
```

6.6.6.7. GET_IDCARD_AGE

This topic describes the GET_IDCARD_AGE function.

Syntax:

```
get_idcard_age(idcardno)
```

Description: This function returns the current age based on the ID card number. The current age is the current year minus the birth year in the ID card number.

Parameters: `idcardno`: the ID card number of the STRING type. The value is a 15-digit or 18-digit number. During data computation, the validity of the ID card is verified based on the province code and the last check code. If the verification fails, NULL is returned.

Return value: A value of the BIGINT type is returned. If the input value is NULL, NULL is returned. If the difference between the current year and the birth year is greater than 100, NULL is returned.

6.6.6.8. GET_IDCARD_BIRTHDAY

This topic describes the GET_IDCARD_BIRTHDAY function.

Syntax:

```
get_idcard_birthday(idcardno)
```

Description: This function returns the date of birth based on the ID card number.

Parameters: `idcardno`: the ID card number of the STRING type. The value is a 15-digit or 18-digit number. During data computation, the validity of the ID card is verified based on the province code and the last check code. If the verification fails, NULL is returned.

Return value: A value of the DATETIME type is returned. If the input value is NULL, NULL is returned.

6.6.6.9. GET_IDCARD_SEX

This topic describes the GET_IDCARD_SEX function.

Syntax:

```
get_idcard_sex(idcardno)
```

Description: This function returns the gender based on the ID card number. The return value is M or F. M indicates male and F indicates female.

Parameters: `idcardno`: the ID card number of the STRING type. The value is a 15-digit or 18-digit number. During data computation, the validity of the ID card is verified based on the province code and the last check code. If the verification fails, NULL is returned.

Return value: A value of the STRING type is returned. If the input value is NULL, NULL is returned.

6.6.6.10. GREATEST

This topic describes the GREATEST function.

Syntax:

```
greatest(var1, var2, ...)
```

Description: This function returns the maximum value of the input parameters.

Parameters: `var1` and `var2`: values of the BIGINT, DOUBLE, DECIMAL, DATETIME, or STRING type. If all input values are NULL, NULL is returned.

Return value:

- The maximum value in the values of all input parameters. If implicit conversion is not required, the data type of the return value is the same as the data types of input parameters.
- NULL is interpreted as the minimum value.

- If a data type conversion is performed among the DOUBLE, BIGINT, and STRING types, a value of the DOUBLE type is returned. If a data type conversion is performed between the STRING and DATETIME types, a value of the DATETIME type is returned. If a data type conversion is performed among the DECIMAL, DOUBLE, BIGINT, and STRING types, a value of the DECIMAL type is returned. Implicit conversions of other data types are not allowed.
- If `odps.sql.hive.compatible` is set to true and the value of an input parameter is NULL, NULL is returned.

6.6.6.11. INDEX

This topic describes the INDEX function and provides examples of using this function.

Syntax:

```
index(var1[var2])
```

Description: This function returns the var2th value if var1 is of the ARRAY<T> type and returns the value whose key is var2 in var1 if var1 is of the MAP<K, V> type.

Parameters:

- var: a value of the ARRAY<T> or MAP<K, V> type.
- var2: If the value of var1 is of the ARRAY<T> type, the value of var2 is of the BIGINT type and greater than or equal to 0. If the value of var1 is of the MAP<K, V> type, the value of var2 is of the K type.

Return value:

- If the value of var1 is of the ARRAY<T> type, a value of the T type is returned. If the value of var2 is beyond the range of elements in the value of the ARRAY<T> type, NULL is returned.
- If the value of var1 is of the MAP<K, V> type, a value of the V type is returned. If the value whose key is var2 does not exist in values of the MAP<K, V> type, NULL is returned.

Examples:

If the value of var1 is an array, execute the following SQL statement:

```
select array('a','b','c')[2] from dual;
-- The following result is returned:
+-----+
| _c0 |
+-----+
| c   |
+-----+
```

If the value of var1 is of the MAP<K, V> type, execute the following SQL statement:

```
select str_to_map("test1=1,test2=2")["test1"] from dual;
-- The following result is returned:
+-----+
| _c0 |
+-----+
| 1   |
+-----+
```

 **Note**

- To execute the SQL statement, remove the index, and execute `var1[var2]` directly. Otherwise, a syntax error is returned.
- If the value of `var1` is NULL, NULL is returned.

6.6.6.12. MAX_PT

This topic describes the MAX_PT function and provides examples of using this function.

Syntax:

```
max_pt(table_full_name)
```

Description: This function returns the maximum value of each first-level partition of a partitioned table in alphabetic order. This function also reads data from the data file in the first-level partitions.

Parameters: `table_full_name`: the table name that has a project name, such as `prj.src`. The value of this parameter is of the STRING type. You must have the permissions to read the table.

Return value: The maximum value in each first-level partition is returned.

Examples:

The `tbl` table is a partitioned table and contains the following partitions:

```
pt = '20120901'  
pt = '20120902'
```

The return value of the following statement is '20120902'. The MaxCompute SQL statement reads data from the 20120902 partition.

```
select * from tbl where pt=max_pt('myproject.tbl');
```

 **Note** If a partition is added by using the ALTER TABLE statement and the partition contains no data file, the maximum value in this partition is not returned.

6.6.6.13. ORDINAL

This topic describes the ORDINAL function and provides examples of using this function.

Syntax:

```
ordinal(bigint nth, var1, var2, ...)
```

Description: This function sorts the input variables in ascending order, and returns the value in the nth bit.

Parameters:

- `nth`: a value of the BIGINT type. This parameter specifies the position in which the value is to be returned. If the value of this parameter is NULL, NULL is returned.
- `var`: the value of the BIGINT, DOUBLE, DATETIME, or STRING type.

Return value:

- The value in the nth bit is returned. If no implicit conversion is required, the return value is of the same data type as the input parameter.

- If a data type conversion is performed among the DOUBLE, BIGINT, and STRING types, a value of the DOUBLE type is returned. If a data type conversion is performed between the STRING and DATETIME types, a value of the DATETIME type is returned. Implicit conversions of other data types are not allowed.
- NULL is interpreted as the minimum value.

Examples:

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) = 2
```

6.6.6.14. LEAST

This topic describes the LEAST function.

Syntax:

```
least(var1, var2, ...)
```

Description: This function returns the minimum value of the input parameters.

Parameters: var: a value of the BIGINT, DOUBLE, DECIMAL, DATETIME, or STRING type. If the values of all input parameters are NULL, NULL is returned.

Return value:

- The minimum value of all values of the input parameters is returned. If implicit conversion is not required, the return value must be of the same data type as the input parameters.
- If a data type conversion is performed among the DOUBLE, BIGINT, and STRING types, a value of the DOUBLE type is returned. If a data type conversion is performed between the STRING and DATETIME types, a value of the DATETIME type is returned. If a data type conversion is performed among the DECIMAL, DOUBLE, BIGINT, and STRING types, a value of the DECIMAL type is returned. Implicit conversions of other data types are not allowed.
- NULL is interpreted as the minimum value.

6.6.6.15. SIZE

This topic describes the SIZE function and provides examples of using this function.

Syntax:

```
INT size(map)  
INT size(array)
```

Description: size(map) returns the number of key-value pairs in the specified map parameter. size(array) returns the number of elements in the specified array parameter.

Parameters:

- map: a value of the MAP type.
- array: a value of the ARRAY type.

Return value: A value of the INT type is returned.

Examples:

```
select size(map('a',123,'b',456));
-- 2 is returned.
select size(map('a',123,'b',456,'c',789));
-- 3 is returned.
select size(array('a','b'));
-- 2 is returned.
select size(array(123,456,789));
-- 3 is returned.
```

6.6.6.16. SPLIT

This topic describes the SPLIT function and provides examples of using this function.

Syntax:

```
split(str, pat)
```

Description: This function returns an array after str is split with pat.

Parameters:

- str: the string that you want to split, which is of the STRING type.
- pat: a delimiter of the STRING type, which supports regular expressions.

Return value: The result after str is split with pat is returned. The return value is of the ARRAY<STRING> type.

Examples:

```
select split("a,b,c",",");
-- The following result is returned:
+-----+
| _c0 |
+-----+
| [a, b, c] |
+-----+
```

6.6.6.17. STR_TO_MAP

This topic describes the STR_TO_MAP function and provides examples of using this function.

Syntax:

```
str_to_map(text [, delimiter1 [, delimiter2]])
```

Description: This function splits text into key-value pairs by using delimiter1 and then splits the key and value in a key-value pair by using delimiter2.

Parameters:

- text: the string that you want to split. The value of this parameter is of the STRING type.
- delimiter1: the delimiter that splits text into key-value pairs. The value of this parameter is of the STRING type. If this parameter is not specified, a comma (,) is used.
- delimiter2: the delimiter that splits each key-value pair into a key and a value. The value of this parameter is of the STRING type. If this parameter is not specified, a period (.) is used.

Return value: The result after text is split by using delimiter1 and delimiter2 is returned. The return value is of the MAP<STRING, STRING> type.

Examples:

```
select str_to_map("test1=1,test2=2");
-- The following result is returned:
+-----+
| a      |
+-----+
| {test1:1, test2:2} |
```

6.6.6.18. UUID

This topic describes the UUID function.

Syntax:

```
string uuid()
```

Description: This function returns a random ID, for example, 29347a88-1e57-41ae-bb68-a9edbdd94212.

 **Note** The return value of this function is a random global ID that has a low probability of duplication.

6.6.6.19. UNIQUE_ID

This topic describes the UNIQUE_ID function.

Syntax:

```
string unique_id()
```

Description: This function returns a unique random ID, for example, 29347a88-1e57-41ae-bb68-a9edbdd94212_1.

 **Note** This function is more efficient than the UUID function.

6.6.6.20. SAMPLE

This topic describes the SAMPLE function and provides examples of using this function.

Syntax:

```
boolean sample(x, y, column_name1, column_name2, ...)
```

Description: This function samples all values that are read from the column specified by column_name based on x and y, and filters out the rows that do not meet the sampling condition.

Parameters:

- x and y: integer constants that are greater than 0. The values of the two parameters are of the BIGINT type. These parameters indicate that the values fall into x portions by calling the hash function and the yth portion is used. If y is not specified, the first portion is used, and you must not specify column_name. If the value of x or y is not of the BIGINT type or is less than or equal to 0, an error is returned. If the value of y is greater than the value of x, an error is returned. If the value of x or y is NULL, NULL is returned.
- column_name: the column from which you want to sample values. This parameter is optional. If this parameter is not specified, random sampling is performed based on the values of x and y. This parameter can be of any data type, and its value can be NULL. No implicit conversion is performed. If the value of column_name is a

constant, an error is returned.

Return value: A value of the BOOLEAN type is returned.

Note To avoid data skew due to NULL values, the system performs uniform hashing on NULL values in the columns specified by `column_name`. This ensures that data evenly falls into `x` portions. If `column_name` is not specified and the amount of data is small, random hashing may be performed. In this case, we recommend that you specify `column_name` for uniform hashing.

Examples:

The `tbla` table contains the `cola` column.

```
select * from tbla where sample (4, 1 , cola) = true;
-- The values in the cola column fall into four portions by calling the hash function, and data in the first portion is extracted.
select * from tbla where sample (4, 2) = true;
-- The values in each row are randomly hashed into four portions, and data in the second portion is extracted.
```

6.6.6.21. CASE WHEN expression

This topic describes the CASE WHEN expression and provides examples of using this function.

MaxCompute provides two syntax formats for the CASE WHEN expression.

```
case value
when value1 then result1
when value2 then result2
...
else resultn
end
```

```
case
when (_condition1) then result1
when (_condition2) then result2
when (_condition3) then result3
...
else resultn
end
```

The CASE WHEN expression can return different values based on the calculation result of the value expression. The following example shows that different regions are obtained based on the value of `shop_name`.

```
select
case
when shop_name is null then 'default_region'
when shop_name like 'hang%' then 'zj_region'
end as region
from sale_detail;
```

Note

- If all result values are of the BIGINT or DOUBLE type, the data types are converted into the DOUBLE type before the values are returned.
- If some result values are of the STRING type, the data types of all values are converted into the STRING type before the values are returned. If the conversion cannot be performed, for example, the BOOLEAN type cannot be converted into the STRING type, an error is returned.
- Conversions between other data types are not allowed.

6.6.6.22. IF

This topic describes the IF function and provides examples of using this function.

Syntax:

```
if(testCondition, valueTrue, valueFalseOrNull)
```

Description: This function determines whether testCondition evaluates to true. If testCondition evaluates to true, valueTrue is returned. Otherwise, valueFalse or NULL is returned.

Parameters:

- testCondition: the expression that you want to evaluate. The value of this parameter is of the BOOLEAN type.
- valueTrue: the value returned if testCondition evaluates to true.
- valueFalseOrNull: the value returned if testCondition evaluates to false. It can be set to NULL.

Return value: The data type of the return value is the same as that of valueTrue or valueFalseOrNull.

Examples:

```
select if(1=2,100,200);
-- The following result is returned:
+-----+
| _c0    |
+-----+
| 200    |
+-----+
```

6.6.6.23. Additional functions

6.6.6.23.1. Usage notes

This topic describes the configuration operation that you must perform before you use other functions that are added to MaxCompute V2.0.

MaxCompute V2.0 provides more other functions. You must add the following SET statement before the SQL statements for these new functions:

```
set odps.sql.type.system.odps2=true;
```



Note You must submit and execute the SET statement and the SQL statements for the new functions at the same time.

6.6.6.23.2. MAP

This topic describes the MAP function and provides examples of using this function.

Syntax:

```
map map(K key1, V value1, K key2, V value2, ...)
```

Description: This function creates mappings by using the given key-value pairs.

Parameters:

- **key:** All keys must be of the same data types, such as those after an implicit conversion. The data types must be basic types.
- **value:** All value types, such as those after an implicit conversion, must be the same and can be of any data types.

Return value: A value of the MAP type is returned.

Examples:

The `t_table` table contains the `c1`, `c2`, `c3`, `c4`, and `c5` columns. The `c1`, `c4`, and `c5` columns are of the `BIGINT` type, and the `c2` and `c3` columns are of the `STRING` type. Data in the `t_table` table:

```
+-----+-----+-----+-----+-----+
| c1      | c2 | c3 | c4      | c5      |
+-----+-----+-----+-----+-----+
| 1000    | k11 | k21 | 86      | 15      |
| 1001    | k12 | k22 | 97      | 2       |
| 1002    | k13 | k23 | 99      | 1       |
+-----+-----+-----+-----+-----+
```

Execute the following statement:

```
select map(c2,c4,c3,c5) from t_table;
-- The following result is returned:
+-----+
| _c0 |
+-----+
| {k11:86, k21:15} |
| {k12:97, k22:2} |
| {k13:99, k23:1} |
+-----+
```

6.6.6.23.3. MAP_KEYS

This topic describes the MAP_KEYS function and provides examples of using this function.

Syntax:

```
ARRAY map_keys(map<K, V>)
```

Description: This function returns all keys in the MAP parameter as an array.

Parameters: `map`: a value of the MAP type.

Return value: A value of the ARRAY type is returned. If the input value is `NULL`, `NULL` is returned.

Examples:

The `t_table_map` table contains two columns: `c1` and `t_map`. `c1` is of the `BIGINT` type and `t_map` is of the `MAP<STRING,BIGINT>` type.

```
+-----+-----+
| c1      | t_map |
+-----+-----+
| 1000    | {k11:86, k21:15} |
| 1001    | {k12:97, k22:2} |
| 1002    | {k13:99, k23:1} |
+-----+-----+
```

Execute the following statement:

```
select c1,map_keys(t_map) from t_table_map;
-- The following result is returned:
+-----+-----+
| c1      | _c1   |
+-----+-----+
| 1000    | [k11, k21] |
| 1001    | [k12, k22] |
| 1002    | [k13, k23] |
+-----+-----+
```

6.6.6.23.4. MAP_VALUES

This topic describes the `MAP_VALUES` function and provides examples of using this function.

Syntax:

```
ARRAY map_values(map<K, V>)
```

Description: This function returns all values in the map parameter as an array.

Parameters: `map`: a value of the `MAP` type.

Return value: A value of the `ARRAY` type is returned. If the input value is `NULL`, `NULL` is returned.

Examples:

```
select map_values(map('a',123,'b',456));
-- The following result is returned:
[123, 456]
```

6.6.6.23.5. SORT_ARRAY

This topic describes the `SORT_ARRAY` function and provides examples of using this function.

Syntax:

```
ARRAY sort_array(ARRAY<T>)
```

Description: This function sorts data in a given array.

Parameters: `ARRAY<T>`: data of the `ARRAY` type. Data in the array can be of any data type.

Return value: A value of the `ARRAY` type is returned.

Examples:

The `t_array` table contains three columns: `c1`, `c2`, and `c3`. The `c1` and `c3` columns are of the `ARRAY<STRING>` type, and the `c2` column is of the `ARRAY<INT>` type.

```
+-----+-----+-----+
| c1      | c2      | c3      |
+-----+-----+-----+
| [a, c, f, b] | [4, 5, 7, 2, 5, 8] | [You, Me, Him] |
+-----+-----+-----+
```

Execute the following statement:

```
select sort_array(c1),sort_array(c2),sort_array(c3) from t_array;
-- The following result is returned:
[a, b, c, f] [2, 4, 5, 5, 7, 8] [Him, You, Me]
```

6.6.6.23.6. POSEXPLODE

This topic describes the `POSEXPLODE` function and provides examples of using this function.

Syntax:

```
posexplode (ARRAY<T>)
```

Description: This function converts a given array into a table that has two columns. The first column lists subscripts of each value in the array, starting from 0. The second column lists array elements.

Parameters: `ARRAY<T>`: a value of the `ARRAY` type. Data in the array can be of any data type.

Return value: The generated table is returned.

Examples:

```
select posexplode(array('a','c','f','b'));
-- The following result is returned:
+-----+-----+
| pos      | val |
+-----+-----+
| 0        | a   |
| 1        | c   |
| 2        | f   |
| 3        | b   |
+-----+-----+
```

6.6.6.23.7. STRUCT

This topic describes the `STRUCT` function and provides examples of using this function.

Syntax:

```
STRUCT struct (value1,value2, ...)
```

Description: This function creates a value of the `STRUCT` type based on the given value list.

Parameters: `value`: a value of any data type.

Return value: A value of the `STRUCT` type is returned. The fields in returned results are sequentially named as `col1`, `col2`,

Examples:

```
select struct('a',123,'ture',56.90);
-- The following result is returned:
{col1:a, col2:123, col3:ture, col4:56.9}
```

6.6.6.23.8. NAMED_STRUCT

This topic describes the NAMED_STRUCT function and provides examples of using this function.

Syntax:

```
STRUCT named_struct(string name1, T1 value1, string name2, T2 value2, ...)
```

Description: This function creates a value of the STRUCT type based on the given name/value list.

Parameters:

- value: a value of any data type.
- name: the name of the field of the STRING type. This parameter is a constant.

Return value: A value of the STRUCT type is returned. Fields in the return value are sequentially named as name1, name2,

Examples:

```
select named_struct('user_id',10001,'user_name','bob','married','F','weight',63.50);
-- The following result is returned:
{user_id:10001, user_name:bob, married:F, weight:63.5}
```

6.6.6.23.9. INLINE

This topic describes the INLINE function and provides examples of using this function.

Syntax:

```
inline(array<STRUCT<f1:T1, f2:T2, ... >>)
```

Description: This function expands a given STRUCT array. Each array element is given one row and each STRUCT element corresponds to one column in each row.

Parameters: STRUCT: the values in the array, which are of any data type.

Return value: The function generated by the table is returned.

Examples:

Column information of the t_table table is t_struct

struct<user_id:bigint,user_name:string,married:string,weight:double>. Data in the t_table table:

```
+-----+
| t_struct |
+-----+
| {user_id:10001, user_name:LiLei, married:N, weight:63.5} |
| {user_id:10002, user_name:HanMeiMei, married:Y, weight:43.5} |
+-----+
```

Execute the following statement:

```
select inline(array(t_struct)) from t_table;
-- The following result is returned:
+-----+-----+-----+-----+
| user_id | user_name | married | weight |
+-----+-----+-----+-----+
| 10001   | LiLei     | N       | 63.5   |
| 10002   | HanMeiMei | Y       | 43.5   |
+-----+-----+-----+-----+
```

6.6.6.23.10. BETWEEN AND expression

This topic describes the BETWEEN AND expression function and provides examples of using this function.

Syntax:

```
A [NOT] BETWEEN B AND C
```

If the value of A, B, or C is NULL, NULL is returned. If the value of A is greater than or equal to the value of B and less than or equal to the value of C, True is returned. Otherwise, False is returned.

Examples:

The emp table contains the following data:

```
| empno | ename | job | mgr | hiredate | sal | comm | deptno |
7369, SMITH, CLERK, 7902, 1980-12-17 00:00:00, 800, , 20
7499, ALLEN, SALESMAN, 7698, 1981-02-20 00:00:00, 1600, 300, 30
7521, WARD, SALESMAN, 7698, 1981-02-22 00:00:00, 1250, 500, 30
7566, JONES, MANAGER, 7839, 1981-04-02 00:00:00, 2975, , 20
7654, MARTIN, SALESMAN, 7698, 1981-09-28 00:00:00, 1250, 1400, 30
7698, BLAKE, MANAGER, 7839, 1981-05-01 00:00:00, 2850, , 30
7782, CLARK, MANAGER, 7839, 1981-06-09 00:00:00, 2450, , 10
7788, SCOTT, ANALYST, 7566, 1987-04-19 00:00:00, 3000, , 20
7839, KING, PRESIDENT, , 1981-11-17 00:00:00, 5000, , 10
7844, TURNER, SALESMAN, 7698, 1981-09-08 00:00:00, 1500, 0, 30
7876, ADAMS, CLERK, 7788, 1987-05-23 00:00:00, 1100, , 20
7900, JAMES, CLERK, 7698, 1981-12-03 00:00:00, 950, , 30
7902, FORD, ANALYST, 7566, 1981-12-03 00:00:00, 3000, , 20
7934, MILLER, CLERK, 7782, 1982-01-23 00:00:00, 1300, , 10
7948, JACCKA, CLERK, 7782, 1981-04-12 00:00:00, 5000, , 10
7956, WELAN, CLERK, 7649, 1982-07-20 00:00:00, 2450, , 10
7956, TEBAGE, CLERK, 7748, 1982-12-30 00:00:00, 1300, , 10
```

Query the data with sal greater than or equal to 1000 and less than or equal to 1500.

```
select * from emp where sal BETWEEN 1000 and 1500;
-- The following result is returned:
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job | mgr | hiredate | sal | comm | deptno |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 7521 | WARD | SALESMAN | 7698 | 1981-02-22 00:00:00 | 1250.0 | 500.0 | 30 |
| 7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 00:00:00 | 1250.0 | 1400.0 | 30 |
| 7844 | TURNER | SALESMAN | 7698 | 1981-09-08 00:00:00 | 1500.0 | 0.0 | 30 |
| 7876 | ADAMS | CLERK | 7788 | 1987-05-23 00:00:00 | 1100.0 | NULL | 20 |
| 7934 | MILLER | CLERK | 7782 | 1982-01-23 00:00:00 | 1300.0 | NULL | 10 |
| 7956 | TEBAGE | CLERK | 7748 | 1982-12-30 00:00:00 | 1300.0 | NULL | 10 |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

6.6.6.23.11. NVL

This topic describes the NVL function and provides an example of using this function.

Function declaration:

```
nvl(T value, T default_value)
```

Description: This function returns default_value if value is NULL. Otherwise, value is returned.

Example:

Assume that the t_data table has three columns: c1 of the STRING type, c2 of the BIGINT type, and c3 of the DATETIME type. The following data is contained in the table:

```
+----+-----+-----+
| c1 | c2      | c3      |
+----+-----+-----+
| NULL | 20      | 2017-11-13 05:00:00 |
| ddd | 25      | NULL     |
| bbb | NULL    | 2017-11-12 08:00:00 |
| aaa | 23      | 2017-11-11 00:00:00 |
+----+-----+-----+
```

After the NVL function is called, the NULL value in c1 is returned as 00000, the NULL value in c2 is returned as 0, and the NULL value in c3 is returned as a hyphen (-).

```
-- Execute the following statement:
select nvl(c1,'00000'),nvl(c2,0) nvl(c3,'-') from nvl_test;
-- Returned result:
+----+-----+-----+
| _c0 | _c1      | _c2 |
+----+-----+-----+
| bbb | 0        | 2017-11-12 08:00:00 |
| ddd | 25      | -   |
| 00000 | 20      | 2017-11-13 05:00:00 |
| aaa | 23      | 2017-11-11 00:00:00 |
+----+-----+-----+
```

6.6.6.23.12. TABLE_EXISTS

This topic describes the TABLE_EXISTS function and provides an example of using this function.

Function declaration:

```
boolean table_exists(string table_name)
```

Description: This function checks whether a specific table exists.

Parameters:

table_name: the table name of the STRING type. The value can include the project name, such as my_proj.my_table. If no project name is specified, the name of the current project is used.

Return value: A value of the BOOLEAN type is returned. If the specified table exists, True is returned. Otherwise, False is returned.

Example:

```
-- Use this function in a SELECT statement.
select if(table_exists('abd'), col1, col2) from src;
```

6.6.6.23.13. PARTITION_EXISTS

This topic describes the PARTITION_EXISTS function and provides an example of using this function.

Function declaration:

```
boolean partition_exists(string table_name, string... partitions)
```

Description: This function checks whether a specific partition exists.

Parameters:

- **table_name**: the table name of the STRING type. The value can include the project name, such as my_proj.my_table. If no project name is specified, the name of the current project is used.
- **partitions**: the partition names of the STRING type. In this parameter, you must specify the values of partition key columns in a table based on the sequence of these columns. The number of values must be the same as the number of partition key columns.

Return value: A value of the BOOLEAN type is returned. If the specified partitions exist, True is returned. Otherwise, False is returned.

Example:

```
create table foo (id bigint) partitioned by (ds string, hr string);
-- Create a partitioned table named foo.
alter table foo add partition (ds='20190101', hr='1');
-- Add a partition to foo.
select partition_exists('foo', '20190101', '1');
-- Check whether partitions with ds='20190101' and hr='1' exist.
```

6.7. UDFs

6.7.1. Overview

This topic describes user-defined functions (UDFs) in MaxCompute.

MaxCompute offers a variety of built-in functions to meet your computing requirements. You can also create UDFs.

UDFs work in a way similar to built-in functions. For the mapping between Java and MaxCompute data types, see [Types of parameters and returned values](#).

Note If a UDF has the same name as a built-in function, MaxCompute executes the UDF by default. To call the built-in function, you must execute the `select ::Function name (expression) ; statement`. For example, if a UDF in MaxCompute is named CONCAT, MaxCompute calls the UDF rather than the CONCAT built-in function. To call the CONCAT built-in function, execute the `select ::concat('ab', 'c') ; statement`.

The following table describes three types of UDFs supported by MaxCompute.

UDF type	Description
----------	-------------

UDF type	Description
UDF	User-defined scalar function. The input and output of a user-defined scalar function have a one-to-one mapping relationship. Every time a user-defined scalar function reads one row of data, one value is returned.
UDTF	User-defined table-valued function. A UDTF is used in scenarios in which multiple rows of data are returned after you call the function. Only this type of function can return multiple fields. A UDTF is not equivalent to a user-defined type (UDT).
UDAF	User-defined aggregate function. The input and output of a user-defined aggregate function have a many-to-one mapping relationship. Multiple input records are aggregated to generate one output value. This type of function can be used with the GROUP BY clause of SQL.

Note When you use a UDF in an SQL statement, the system may prompt insufficient memory. The reason is that the memory size required for the computing task exceeds the default memory size. In this case, you can execute the `set odps.sql.udf.joiner.jvm.memory=xxxx;` statement to adjust the memory size.

MaxCompute UDFs can be shared by multiple projects. UDFs in one project can be used in another project. SQL statement for cross-project UDF sharing:

```
select other_project:udf_in_other_project(arg0, arg1) as res from table_t;
```

If you use Maven, you can obtain the dependency from the [Maven repository](#).

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.29.10-public</version>
</dependency>
```

6.7.2. Types of parameters and return values

This topic describes the types of parameters and return values of user-defined functions (UDFs) in MaxCompute.

Java UDFs in MaxCompute V2.0 support more basic data types in addition to BIGINT, STRING, DOUBLE, and BOOLEAN. These UDFs also support complex data types, such as ARRAY, MAP, and STRUCT, as well as Writable types.

To use a new basic data type, UDFs specify their function signatures in the following ways:

- A user-defined aggregate function (UDAF) or user-defined table-valued function (UDTF) uses the @Resolve annotation, for example, `@Resolve("smallint->varchar(10)")`.
- A user-defined scalar function uses the evaluate method. In this case, the built-in function types of MaxCompute and Java function types have a one-to-one mapping relationship.

 Notice

- You can use `type, *` to pass any number of parameters, for example, `@resolve("string,*->array<string>")`. You must add Subtype after ARRAY.
- The field name and field type of `com.aliyun.odps.data.Struct` cannot be reflected. Therefore, you must use the `@Resolve` annotation to obtain the field name and field type. In other words, to use STRUCT in a UDF, you must add the `@Resolve` annotation to the UDF class. This annotation affects only the overloads of parameters or return values that contain `com.aliyun.odps.data.Struct`.
- Only one `@Resolve` annotation can be provided for the class. Therefore, only one overload with a STRUCT parameter or return value exists in a UDF.

To use a complex data type, UDFs specify their function signatures in the following ways:

- A UDTF uses the `@Resolve` annotation, for example, `@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")`.
- A user-defined scalar function uses the signature of the evaluate method to match input and output types. This involves the following mapping between MaxCompute and Java data types:
 - ARRAY in MaxCompute maps to `java.util.List`.
 - MAP in MaxCompute maps to `java.util.Map`.
 - STRUCT in MaxCompute maps to `com.aliyun.odps.data.Struct`.
- A UDAF uses the `@Resolve` annotation, for example, `@Resolve("smallint->varchar(10)")`.

The following table describes the mapping between MaxCompute and Java data types.

MaxCompute data type	Java data type
TINYINT	<code>java.lang.Byte</code>
SMALLINT	<code>java.lang.Short</code>
INT	<code>java.lang.Integer</code>
BIGINT	<code>java.lang.Long</code>
FLOAT	<code>java.lang.Float</code>
DOUBLE	<code>java.lang.Double</code>
DECIMAL	<code>java.math.BigDecimal</code>
BOOLEAN	<code>java.lang.Boolean</code>
STRING	<code>java.lang.String</code>
VARCHAR	<code>com.aliyun.odps.data.Varchar</code>
BINARY	<code>com.aliyun.odps.data.Binary</code>
DATETIME	<code>java.util.Date</code>
TIMESTAMP	<code>java.sql.Timestamp</code>
ARRAY	<code>java.util.List</code>
MAP	<code>java.util.Map</code>

MaxCompute data type	Java data type
STRUCT	com.aliyun.odps.data.Struct

 **Note**

- Make sure that the input and output parameters in a UDF are of a Java type. Otherwise, error ODPS-0130071 is returned.
- Java data types and the data types of return values are objects. Java data types must start with an uppercase letter.
- The NULL value in SQL statements is represented by a NULL reference in Java. Java Primitive Type cannot have NULL values and must not be used.
- The ARRAY type in the preceding table maps java.util.List.

MaxCompute V2.0 allows you to use Writable types as parameters and return values when you define Java UDFs. The following table describes the mapping between MaxCompute data types and Java Writable types.

MaxCompute data type	Java Writable type
TINYINT	ByteWritable
SMALLINT	ShortWritable
INT	IntWritable
BIGINT	LongWritable
FLOAT	FloatWritable
DOUBLE	DoubleWritable
DECIMAL	BigDecimalWritable
BOOLEAN	BooleanWritable
STRING	Text
VARCHAR	VarcharWritable
BINARY	BytesWritable
DATETIME	DatetimeWritable
TIMESTAMP	TimestampWritable
INTERVAL_YEAR_MONTH	IntervalYearMonthWritable
INTERVAL_DAY_TIME	IntervalDayTimeWritable
ARRAY	N/A
MAP	N/A
STRUCT	N/A

6.7.3. UDF

This topic describes user-defined scalar functions (UDFs) and provides examples of using UDFs.

UDFs must inherit the class `com.aliyun.odps.udf.UDF` and implement the `evaluate` method of the class. The `evaluate` method must be a non-static public method, and the types of parameters and return values are used as the UDF signature in SQL statements. You can implement multiple `evaluate` methods in a UDF. To call a UDF, the framework matches the correct `evaluate` method based on the parameter type called by the UDF.

Note

- We recommend that you do not use the classes that have the same name but different function logic in different JAR packages. For example, in UDF(UDAF/UDTF): `udf1`, `udf2`, the JAR package of `udf1` is `udf1.jar` and the JAR package of `udf2` is `udf2.jar`. Assume that the two JAR packages contain the `com.aliyun.UserFunction.class` class. If you use the two UDFs in the same SQL statement, the system randomly loads the class contained in one of the two JAR packages. This may cause the UDFs to be executed in different ways or even cause a compilation failure.
- If you execute an SQL statement that has two UDFs, the UDFs are not isolated from each other. This is because the UDFs share the same classpath. If the resources referenced by the UDFs contain the same class, the class that the classloader attempts to load is uncertain. To avoid this issue, make sure that the resources referenced by the two UDFs do not contain the same class.

You can use `void setup(ExecutionContext ctx)` to initialize a UDF and use `void close()` to terminate a UDF.

The following example shows how to implement a UDF:

```
package org.alidata.odps.udf.examples;
import com.aliyun.odps.udf.UDF;
public final class Lower extends UDF {
    public String evaluate(String s) {
        if (s == null) {
            return null;
        }
        return s.toLowerCase();
    }
}
```

The following example shows how to implement Concat by using a Writable type:

```
package com.aliyun.odps.udf.example;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
public class MyConcat extends UDF {
    private Text ret = new Text();
    public Text evaluate(Text a, Text b) {
        if (a == null || b == null) {
            return null;
        }
        ret.clear();
        ret.append(a.getBytes(), 0, a.getLength());
        ret.append(b.getBytes(), 0, b.getLength());
        return ret;
    }
}
```

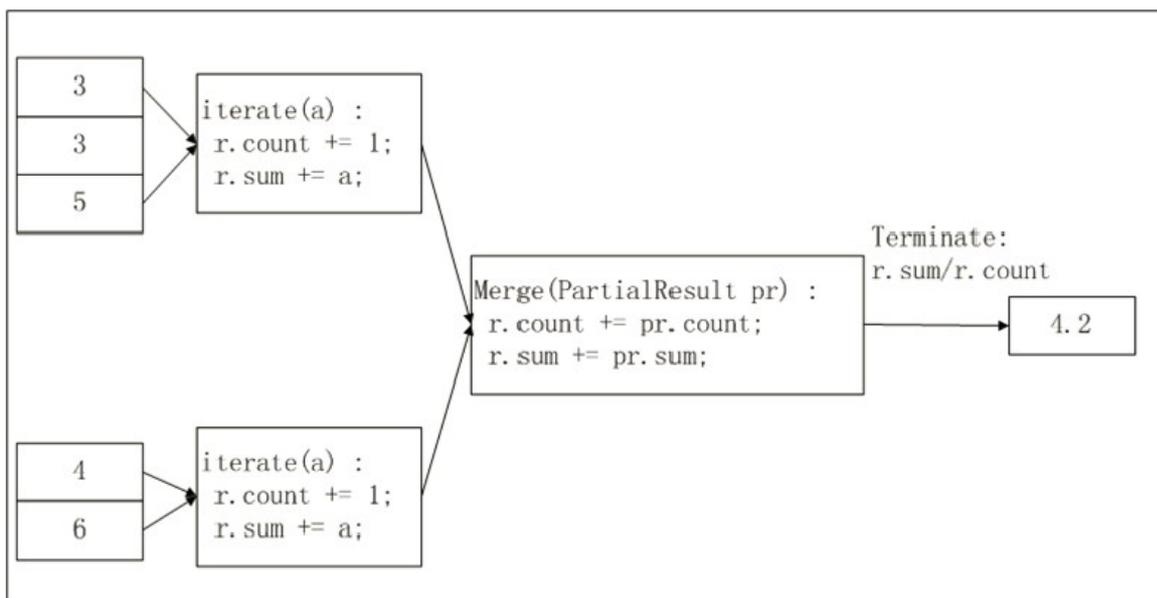
6.7.4. UDAF

This topic describes user-defined aggregate functions (UDAFs) and provides examples of using UDAFs.

UDAFs must inherit the `com.aliyun.odps.udf.Aggregator` class and implement the following methods:

```
import com.aliyun.odps.udf.ContextFunction;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
public abstract class Aggregator implements ContextFunction {
    @Override
    public void setup(ExecutionContext ctx) throws UDFException {
    }
    @Override
    public void close() throws UDFException {
    }
    /**
     * Create an aggregation buffer * @return Writable aggregation buffer
     */
    abstract public Writable newBuffer();
    /**
     * @param buffer: aggregation buffer * @param args: a parameter specified to call a UDAF in SQL.
     It cannot be NULL, but the values in args can be NULL, which indicates that the input data is NULL *
     @throws UDFException.
     */
    abstract public void iterate(Writable buffer, Writable[] args) throws UDFException;
    /**
     * Generate the final result * @param buffer * @return Final result of Object UDAF * @throws UDF
     Exception.
     */
    abstract public Writable terminate(Writable buffer) throws UDFException;
    abstract public void merge(Writable buffer, Writable partial) throws UDFException;
}
```

The most important methods are `iterate`, `merge`, and `terminate` because the main logic of UDAFs relies on these methods. In addition, you must implement the user-defined `Writable` buffer. The following figure illustrates the implementation logic and procedure to calculate the avg value by using a MaxCompute UDAF.



In the preceding figure, the input data is sliced based on the specified size. The size of each slice is suitable for a worker to complete the calculation in a specified time. The slice size must be manually configured.

The calculation procedure of a UDAF involves two steps:

- Step 1: Each worker counts the data quantity and total sum in a slice. You can consider the data quantity and total sum in each slice as an intermediate result.
- Step 2: Each worker gathers the information about each slice generated in Step 1. In the final output, $r.sum/r.count$ is the average value of all input data.

Sample code for a UDAF that is used to calculate the average value:

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("double->double")
public class AggrAvg extends Aggregator {
    private static class AvgBuffer implements Writable {
        private double sum = 0;
        private long count = 0;
        @Override
        public void write(DataOutput out) throws IOException {
            out.writeDouble(sum);
            out.writeLong(count);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            sum = in.readDouble();
            count = in.readLong();
        }
    }
    private DoubleWritable ret = new DoubleWritable();
    @Override
    public Writable newBuffer() {
        return new AvgBuffer();
    }
    @Override
    public void iterate(Writable buffer, Writable[] args) throws UDFException {
        DoubleWritable arg = (DoubleWritable) args[0];
        AvgBuffer buf = (AvgBuffer) buffer;
        if (arg != null) {
            buf.count += 1;
            buf.sum += arg.get();
        }
    }
    @Override
    public Writable terminate(Writable buffer) throws UDFException {
        AvgBuffer buf = (AvgBuffer) buffer;
        if (buf.count == 0) {
            ret.set(0);
        } else {
            ret.set(buf.sum / buf.count);
        }
        return ret;
    }
    @Override
    public void merge(Writable buffer, Writable partial) throws UDFException {
        AvgBuffer buf = (AvgBuffer) buffer;
        AvgBuffer p = (AvgBuffer) partial;
        buf.sum += p.sum;
        buf.count += p.count;
    }
}
```

Sample code for implementing Concat by using Writable types:

```

package com.aliyun.odps.udf.example;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
public class MyConcat extends UDF {
    private Text ret = new Text();
    public Text evaluate(Text a, Text b) {
        if (a == null || b == null) {
            return null;
        }
        ret.clear();
        ret.append(a.getBytes(), 0, a.getLength());
        ret.append(b.getBytes(), 0, b.getLength());
        return ret;
    }
}

```

6.7.5. UDTFs

6.7.5.1. Overview

This topic describes user-defined table-valued functions (UDTFs) and provides examples of using UDTFs.

UDTFs must inherit the `com.aliyun.odps.udf.UDTF` class and implement four methods. The following table describes the methods.

Method	Description
<code>public void setup(ExecutionContext ctx) throws UDFException</code>	The initialization method to call the user-defined initialization behavior before a UDTF processes the input data. <code>setup</code> is called once first for each worker.
<code>public void process(Object[] args) throws UDFException</code>	This method is called by the framework. Each SQL record calls <code>process</code> once. The parameters of <code>process</code> are the input parameters of the UDTF specified in the SQL statement. The input parameters are passed in as <code>Object[]</code> , and the results are returned by using the <code>forward</code> function. You must call <code>forward</code> in the <code>process</code> function to determine the output data.
<code>public void close() throws UDFException</code>	The method for terminating a UDTF. The framework calls this method only once. The method is called after the last record is processed.
<code>public void forward(Object ...o) throws UDFException</code>	You can call the <code>forward</code> method to return data. One record is returned each time <code>forward</code> is called. The record corresponds to the column specified by the <code>AS</code> clause in the SQL statement of the UDTF.

Example:

```

package org.alidata.odps.udtf.examples;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDTFCollector;
import com.aliyun.odps.udf.annotation.Resolve;
import com.aliyun.odps.udf.UDFException;
// TODO define input and output types, e.g., "string,string->string,bigint".
@Resolve("string,bigint->string,bigint")
public class MyUDTF extends UDTF {
    @Override
    public void process(Object[] args) throws UDFException {
        String a = (String) args[0];
        Long b = (Long) args[1];
        for (String t: a.split("\\s+")) {
            forward(t, b);
        }
    }
}

```

Assume that you create a UDTF in MaxCompute with the registered function name of `user_udtf`. Execute the following statement to use the UDTF:

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
```

The following example shows the values of `col0` and `col1` in `my_table`.

```

+-----+-----+
| col0 | col1 |
+-----+-----+
| A B  | 1    |
| C D  | 2    |
+-----+-----+

```

The following example shows the execution result of the `SELECT` statement.

```

+----+----+
| c0 | c1 |
+----+----+
| A  | 1  |
| B  | 1  |
| C  | 2  |
| D  | 2  |
+----+----+

```

6.7.5.2. Examples of using UDTFs

This topic provides examples of using UDTFs.

Use a UDTF to read resources in MaxCompute

You can use a UDTF to read resources in MaxCompute. Example:

1. Compile a UDTF program. After the compilation is successful, export the JAR package. In this example, the JAR package is named `udtfexample1.jar`.

```

package com.aliyun.odps.examples.udf;
import java.io.BufferedReader;
import java.io.IOException;

```

```

import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Iterator;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.annotation.Resolve;
/**
 * project: example_project
 * table: wc_in2
 * partitions: p2=1,p1=2
 * columns: colc,colb
 */
@Resolve("string,string->string,bigint,string")
public class UDTFResource extends UDTF {
    ExecutionContext ctx;
    long fileResourceLineCount;
    long tableResource1RecordCount;
    long tableResource2RecordCount;
    @Override
    public void setup(ExecutionContext ctx) throws UDFException {
        this.ctx = ctx;
        try {
            InputStream in = ctx.readResourceFileAsStream("file_resource.txt");
            BufferedReader br = new BufferedReader(new InputStreamReader(in));
            String line;
            fileResourceLineCount = 0;
            while ((line = br.readLine()) != null) {
                fileResourceLineCount++;
            }
            br.close();
            Iterator<Object[]> iterator = ctx.readResourceTable("table_resource1").iterator();
            tableResource1RecordCount = 0;
            while (iterator.hasNext()) {
                tableResource1RecordCount++;
                iterator.next();
            }
            iterator = ctx.readResourceTable("table_resource2").iterator();
            tableResource2RecordCount = 0;
            while (iterator.hasNext()) {
                tableResource2RecordCount++;
                iterator.next();
            }
        } catch (IOException e) {
            throw new UDFException(e);
        }
    }
    @Override
    public void process(Object[] args) throws UDFException {
        String a = (String) args[0];
        long b = args[1] == null ? 0 : ((String) args[1]).length();
        forward(a, b, "fileResourceLineCount=" + fileResourceLineCount + "|tableResource1RecordCount="
+ tableResource1RecordCount + "|tableResource2RecordCount=" + tableResource2RecordCount);
    }
}

```

2. Add resources to MaxCompute.

```
Add file file_resource.txt;
Add jar udtfexample1.jar;
Add table table_resource1 as table_resource1;
Add table table_resource2 as table_resource2;
```

3. Create the my_udtf UDF in MaxCompute.

```
create function mp_udtf as com.aliyun.odps.examples.udf.UDTFResource using
'udtfexample1.jar, file_resource.txt, table_resource1, table_resource2';
```

4. Run this UDF.

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from table_resource1;
```

Returned result:

```
+-----+-----+-----+
| a | b      | fileResourceLineCount |
+-----+-----+-----+
| 10 | 2      | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordC
ount=0 |
| 10 | 2      | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordC
ount=0 |
+-----+-----+-----+
```

Use complex data types in a UDF

The following example defines a UDF with three overloads. The first overload uses the ARRAY type, the second overload uses the MAP type, and the third overload uses the STRUCT type. The third overload uses STRUCT as the data type of parameters or returned values. Therefore, the @Resolve annotation must be added to the UDF class to specify the STRUCT type.

```
import com.aliyun.odps.udf.UDF;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("struct,string->string")
public class UdfArray extends UDF {
    public String evaluate(List vals, Long len) {
        return vals.get(len.intValue());
    }
    public String evaluate(Map map, String key) {
        return map.get(key);
    }
    public String evaluate(Struct struct, String key) {
        return struct.getFieldValue("a") + key;
    }
}
```

You can pass complex data types into a UDF.

```
create function my_index as 'UdfArray' using 'myjar.jar';
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as color_name from co
```

Use Hive UDFs

MaxCompute V2.0 supports Hive UDFs. Some Hive UDFs and UDTFs can be directly used in MaxCompute.

Note MaxCompute V2.0 is compatible with Hive 2.1.0 and Hadoop 2.7.2. If you develop a UDF by using other versions of Hive or Hadoop, you may need to recompile the UDF on the compatible Hive or Hadoop version.

The following example shows how to use a Hive UDF in MaxCompute:

```
package com.aliyun.odps.compiler.hive;
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
public class Collect extends GenericUDF {
    @Override
    public ObjectInspector initialize(ObjectInspector[] objectInspectors) throws UDFArgumentException
    {
        if (objectInspectors.length == 0) {
            throw new UDFArgumentException("Collect: input args should >= 1");
        }
        for (int i = 1; i < objectInspectors.length; i++) {
            if (objectInspectors[i] != objectInspectors[0]) {
                throw new UDFArgumentException("Collect: input oi should be the same for all args");
            }
        }
        return ObjectInspectorFactory.getStandardListObjectInspector(objectInspectors[0]);
    }
    @Override
    public Object evaluate(DeferredObject[] deferredObjects) throws HiveException {
        List<Object> objectList = new ArrayList<>(deferredObjects.length);
        for (DeferredObject deferredObject : deferredObjects) {
            objectList.add(deferredObject.get());
        }
        return objectList;
    }
    @Override
    public String getDisplayString(String[] strings) {
        return "Collect";
    }
}
```

The Hive UDF in the example packages any type and number of parameters into arrays. In this example, the output JAR package is named test.jar.

```
-- Add a resource.
Add jar test.jar;
-- Create a Hive UDF.
CREATE FUNCTION hive_collect as 'com.aliyun.odps.compiler.hive.Collect' using 'test.jar';
-- Use the Hive UDF.
set odps.sql.hive.compatible=true;
select hive_collect(4y,5y,6y) from dual;
```

Returned result:

```
+-----+
| _c0 |
+-----+
| [4, 5, 6] |
+-----+
```

Note Take note of the following items when you use Hive UDFs that are compatible with MaxCompute:

- Specify the JAR package when you create a Hive UDF. MaxCompute cannot automatically add all JAR packages to the classpath. The `add jar` command in MaxCompute creates a permanent resource in the project.
- Insert `set odps.sql.hive.compatible=true;` before an SQL statement, and commit and execute the SET statement with the SQL statement.
- Pay attention to the limits imposed by the Java sandbox on MaxCompute.

6.7.6. Python UDFs

6.7.6.1. Before you start

This topic describes the prerequisites for the use of Python user-defined functions (UDFs).

Before you use Python UDFs, you must enable the isolation sandbox for a specified project. For more information about how to enable an isolation sandbox, contact technical support.

6.7.6.2. Limits

This topic describes limits on Python user-defined functions (UDFs).

MaxCompute uses Python 2.7. User code runs in a sandbox, which is a limited environment. In this environment, the following operations are prohibited:

- Read and write local files.
- Start subprocesses.
- Start threads.
- Initiate socket-based communication.
- Call other systems.

Due to these limits, the code that you upload must be implemented by standard Python, and C extension modules are not allowed in your code.

In addition, the modules that involve the preceding operations are disabled in the Python standard library. The following modules are available in the standard library:

- The modules that are implemented by standard Python.
- The following C extension modules:
 - array
 - audioop
 - binascii
 - _bisect
 - cmath
 - _codecs_cn
 - _codecs_hk

- `_codecs_iso2022`
- `_codecs_jp`
- `_codecs_kr`
- `_codecs_tw`
- `_collections`
- `cStringIO`
- `datetime`
- `_functools`
- `future_builtins`
- `_hashlib`
- `_heapq`
- `itertools`
- `_json`
- `_locale`
- `_lsprof`
- `math`
- `_md5`
- `_multibytecodec`
- `operator`
- `_random`
- `_sha256`
- `_sha512`
- `_sha`
- `_struct`
- `strop`
- `time`
- `unicodedata`
- `_weakref`
- `cPickle`

- Some modules with limited functionality. For example, the size of data that your code can write into standard output `sys.stdout` and standard error output `sys.stderr` in a sandbox is limited to 20 KB. Characters that exceed this limit are ignored.

6.7.6.3. Third-party libraries

This topic describes third-party libraries for Python user-defined functions (UDFs).

Common third-party libraries such as NumPy are installed in the runtime environment to supplement the standard library.

 **Warning** The use of third-party libraries is also subject to limits. For example, when you use a third-party library, you are not allowed to access local data and you can use only limited network I/O resources. The related APIs in the third-party libraries are disabled.

6.7.6.4. Types of parameters and return values

This topic describes the types of parameters and return values of Python user-defined functions (UDFs).

The types of parameters and return values are specified by using the following method:

```
@odps.udf.annotate(signature)
```

Python UDFs support MaxCompute SQL data types, such as BIGINT, STRING, DOUBLE, BOOLEAN, DATETIME, DECIMAL, complex data types, and nested complex data types. Complex data types include ARRAY, MAP, and STRUCT. Before you execute an SQL statement, the types of parameters and return values of all functions must be determined. Python is a dynamically typed language. You must add decorators to the UDF class to specify the function signature.

The function signature is specified by a string. The following syntax is supported:

```
arg_type_list '->' type_list
arg_type_list: type_list | '*' | '' | 'char(n)' | 'varchar(n)'
type_list: [type_list ','] type
type: 'bigint' | 'string' | 'double' | 'boolean' | 'datetime' | 'float' | 'binary' | 'date' | 'decimal' | 'decimal(precision,scale)'
```

Note

- The part to the left of the arrow indicates the types of parameters. The part to the right of the arrow indicates the types of return values.
- The return value of a UDTF can contain multiple columns, while the return value of a UDF or UDAF contains only one column.
- An asterisk (*) represents variable-length parameters. If variable-length parameters are used, UDFs, UDTFs, and UDAFs can match input parameters of any type.
- In a project that uses the MaxCompute V2.0 data type edition, you can set the decimal parameter to precision or scale.
- The return value cannot be of the CHAR or VARCHAR type.

The following example shows a valid signature:

```
'bigint,double->string' # Parameters are of the BIGINT and DOUBLE types, and return values are of the STRING type.
'bigint,boolean->string,datetime' # UDTF parameters are of the BIGINT and BOOLEAN types, and return values are of the STRING and DATETIME types.
'*->string' # Input parameters are variable-length parameters and can be of any type, and return values are of the STRING type.
'->double' # Parameters are left empty, and return values are of the DOUBLE type.
'array<bigint>->struct<x:string, y:int>' # Parameters are of the ARRAY<BIGINT> type, and return values are of the STRUCT<x:STRING, y:BIGINT> type.
'->map<bigint, string>' # Parameters are left empty, and return values are of the MAP<BIGINT, STRING> type.
```

If an invalid signature is found during query parsing, an error is reported, and the execution of the function is prohibited. During the execution of the function, the UDF parameters of the type specified by the function signature are passed. The return values must be of the same type as those specified by the function signature. Otherwise, an error is reported. The following table describes the mappings between Python data types and MaxCompute SQL data types.

MaxCompute SQL data type	Python data type
BIGINT	INT

MaxCompute SQL data type	Python data type
STRING	STR
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	INT
FLOAT	FLOAT
CHAR	STR
VARCHAR	STR
BINARY	BYTEARRAY
DATE	INT
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICTIONARY
STRUCT	COLLECTIONS.NAMEDTUPLE

Note

- A value of the DATETIME type is converted to a value of the INT type. The value of the INT type follows the UNIX format, which is the number of milliseconds that have elapsed since 00:00:00 Thursday, January 1, 1970. You can process data of the DATETIME type by using the DATETIME module in the Python standard library.
- silent is added to `odps.udf.int(value, [silent=True])`. If silent is set to True and value cannot be converted to the INT type, None is returned, and no error is reported.
- The NULL value corresponds to None in Python.

6.7.6.5. UDF

This topic describes user-defined scalar functions (UDFs) and provides an example of using UDFs.

To implement Python UDFs, you must define a new-style class and implement the evaluate method.

```
from odps.udf import annotate
@annotate("bigint,bigint->bigint")
class MyPlus(object):
    def evaluate(self, arg0, arg1):
        if None in (arg0, arg1):
            return None
        return arg0 + arg1
```

 **Note** A Python UDF must have its signature specified by using annotate.

6.7.6.6. UDAF

This topic describes user-defined aggregate functions (UDAFs) and provides an example of using UDAFs.

- `class odps.udf.BaseUDAF`: is inherited to implement Python UDAFs.
- `BaseUDAF.new_buffer()`: returns buffer of the intermediate value of a UDAF. buffer must be a marshallable object (such as LIST or DICT), and the buffer size cannot increase with the data volume. Under extreme circumstances, the buffer size cannot exceed 2 MB after the marshal operation.
- `BaseUDAF.iterate(buffer[, args, ...])`: aggregates args to buffer of the intermediate value.
- `BaseUDAF.merge(buffer, pBuffer)`: aggregates buffer of two intermediate values. It merges pBuffer to buffer.
- `BaseUDAF.terminate(buffer)`: converts buffer of the intermediate value to a value of a basic data type in MaxCompute SQL.

Example of using a UDAF to obtain the average value:

```
@annotate('double->double')
class Average(BaseUDAF):
    def new_buffer(self):
        return [0, 0]
    def iterate(self, buffer, number):
        if number is not None:
            buffer[0] += number
            buffer[1] += 1
    def merge(self, buffer, pBuffer):
        buffer[0] += pBuffer[0]
        buffer[1] += pBuffer[1]
    def terminate(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

6.7.6.7. UDTF

This topic describes user-defined table-valued functions (UDTFs) and provides an example of using UDTFs.

- `class odps.udf.BaseUDTF`: the base class of Python UDTFs. This class can be inherited to implement the PROCESS and CLOSE methods.
- `BaseUDTF.__init__()`: the initialization method. To implement this method for a derived class, you must call the `super(BaseUDTF, self).__init__()` initialization method of the base class at the beginning of code execution. Throughout the lifecycle of a UDTF, the INIT method is called only once. It is called only before the first record is processed. If a UDTF needs to save internal states, all states can be initialized by using this method.
- `BaseUDTF.process([args, ...])`: is called by the MaxCompute SQL framework. The PROCESS method is called for each record in SQL. The parameters in the PROCESS method are the UDTF input parameters that are specified in SQL statements.
- `BaseUDTF.forward([args, ...])`: the UDTF output method. This method is called by user code. One record is generated each time the FORWARD method is called. The parameters in the FORWARD method are the UDTF output parameters that are specified in SQL statements.
- `BaseUDTF.close()`: the UDTF termination method. This method is called only once by the MaxCompute SQL framework. It is called only after the last record is processed.

Example:

```
#coding:utf-8
# explode.py
from odps.udf import annotate
from odps.udf import BaseUDTF
@annotate('string -> string')
class Explode(BaseUDTF):
    # Use commas (,) to separate the string into multiple records.
    def process(self, arg):
        props = arg.split(',')
        for p in props:
            self.forward(p)
```

 **Note** The types of parameters and the return value of a Python UDF can be specified without the need to use `annotate` to specify the UDF signature. This way, the function can match input parameters of any type in SQL. However, the type of the return value cannot be deduced. All output parameters are considered to be of the `STRING` type. Therefore, when `FORWARD` is called, all output values must be converted to the `STRING` type.

6.7.6.8. Resource reference

This topic describes how to reference resources in Python user-defined functions (UDFs) and provides an example for reference.

You can reference file and table resources in Python UDFs by using the `odps.distcache` module.

The following sample code shows the syntax that is used to reference file resources:

```
odps.distcache.get_cache_file(resource_name)
-- Return the content of a specified resource.
```

The `resource_name` parameter is a string that corresponds to the name of an existing file resource in the current project. If the resource name is invalid or the file resource does not exist, an error is returned.

The return value is a file-like object. If this object is no longer used, the caller must call the `close` method to release the open file resource.

Example:

```
@annotate('bigint->string')
class DistCacheExample(object):
    def __init__(self):
        cache_file = get_cache_file('test_distcache.txt')
        kv = {}
        for line in cache_file:
            line = line.strip()
            if not line:
                continue
            k, v = line.split()
            kv[int(k)] = v
        cache_file.close()
        self.kv = kv
    def evaluate(self, arg):
        return self.kv.get(arg)
```

The following sample code shows the syntax that is used to reference table resources:

```
odps.distcache.get_cache_table(resource_name)
```

The `resource_name` parameter is a string that corresponds to the name of an existing table resource in the current project. If the resource name is invalid or the table resource does not exist, an error is returned.

The return value is of the GENERATOR type. The caller traverses the table to obtain the content. A record of the ARRAY type is obtained each time the caller traverses the table.

Example:

```
from odps.udf import annotate
from odps.distcache import get_cache_table
@annotate('->string')
class DistCacheTableExample(object):
    def __init__(self):
        self.records = list(get_cache_table('udf_test'))
        self.counter = 0
        self.ln = len(self.records)
    def evaluate(self):
        if self.counter > self.ln - 1:
            return None
        ret = self.records[self.counter]
        self.counter += 1
        return str(ret)
```

6.7.7. Python 3 UDF

This topic describes the precautions when you use Python 3 user-defined functions (UDFs).

Python Software Foundation announces the End of Life (EOL) for Python 2. Due to this reason, MaxCompute supports Python 3 and uses Python 3.7.3.

Limits

Python 3 is not compatible with Python 2. You can select Python 2 or Python 3 to execute an SQL statement. However, you cannot use Python 2 code and Python 3 code in a single SQL statement at the same time.

Enable Python 3

You can enable Python 3 only for jobs. To enable Python 3 for a job, add the following command before the required SQL statement and execute them together:

```
set odps.sql.python.version=cp37;
```

Use third-party libraries

MaxCompute Python UDFs support third-party libraries. The third-party library NumPy is installed in the Python 2 runtime environment to supplement the standard library.

NumPy is not installed in the Python 3 runtime environment in MaxCompute. To use a NumPy UDF, you must manually upload a NumPy wheel package. If you download a NumPy wheel package from [Python Package Index \(PyPI\)](#), the package name is `numpy-<Version>-cp37-cp37m-manylinux1_x86_64.whl`.

Port Python 2 UDFs

Python Software Foundation announces the EOL for Python 2. Therefore, we recommend that you port Python 2 UDFs. The method used to port Python 2 UDFs varies based on projects.

- In a new project or an existing project where no Python UDFs exist, we recommend that you use Python 3 to

write all Python UDFs.

- Exercise caution when you enable Python 3 in an existing project where a large number of Python 2 UDFs exist. This is because you cannot use Python 2 code and Python 3 code in a single SQL statement at the same time. If you plan to gradually replace Python 2 UDFs with Python 3 UDFs, use the following methods:
 - Use Python 3 to write new UDFs and enable Python 3 for new jobs.
 - Rewrite Python 2 UDFs to make them compatible with both Python 2 and Python 3. For more information about how to rewrite UDFs, see [Porting Python 2 code to Python 3](#).

Note If you write a public UDF that needs to be shared among multiple projects, we recommend that you write the UDF to be compatible with both Python 2 and Python 3.

6.7.8. SQL Function

This topic describes how to use SQL functions. SQL functions allow you to reference SQL user-defined functions (UDFs) in SQL scripts.

Background information

You can use SQL functions of MaxCompute to resolve the following issues:

- Generally, a large amount of similar code exists, which is inconvenient to maintain and prone to errors. To use UDFs, you must develop code, compile the code in Java, and then create resources and functions. The process is complicated. In addition, UDFs cannot catch up with built-in functions in terms of performance. Example:

```
SELECT
  NVL(STR_TO_MAP(GET_JSON_OBJECT(col, '$.key1')), 'default') AS key1,
  NVL(STR_TO_MAP(GET_JSON_OBJECT(col, '$.key2')), 'default') AS key2,
  ...
  NVL(STR_TO_MAP(GET_JSON_OBJECT(col, '$.keyN')), 'default') AS keyN
FROM t;
```

- One function can be transferred as a parameter to another only by using code similar to lambda expressions in Java.

Feature description

As a type of UDFs, SQL functions allow you to create UDFs in SQL and use function type parameters and anonymous functions. This way, you can define business logic more flexibly. You can use SQL functions to simplify feature implementation and improve code reuse. SQL functions provide the following features:

- SQL functions allow you to reference and call SQL UDFs in SQL scripts.
- You can specify built-in functions, UDFs, or SQL functions in function type parameters when you call SQL functions.
- You can specify anonymous functions in function type parameters when you call SQL functions.

Create a permanent SQL function

After you create a permanent SQL function and store it in the metadata system, all query statements can reference this function. To reference SQL functions, you must have function-level permissions. The following example shows the syntax:

```
CREATE SQL FUNCTION function_name(@parameter_in1 datatype[, @parameter_in2 datatype...])
[RETURNS @parameter_out datatype]
AS [BEGIN]
function_expression
[END];
```

Parameters:

- `function_name`: the name of the SQL function that you want to create. Each function name must be unique and can be registered only once. The SQL function names cannot be the same as those of built-in functions.
- `parameter_in`: the input parameters of the SQL function.
- `RETURNS`: the variable to be returned by the SQL function. If you do not specify `RETURNS`, the value of `function_name` is returned.
- `parameter_out`: the output parameters of the SQL function.
- `function_expression`: the expression of the SQL function.

Example:

```
CREATE SQL FUNCTION MY_ADD(@a BIGINT) AS @a + 1;
```

In the preceding example, `@a + 1` indicates the implementation logic of the SQL function. You can write it as an expression to specify a built-in operator, built-in function, or UDF.

If the implementation logic is complex, you can write multiple SQL statements and enclose them with `BEGIN` and `END`. `RETURNS` specifies the variable to be returned by the SQL function. If you do not specify `RETURNS`, the value of `function_name` is returned.

Example:

```
CREATE SQL FUNCTION MY_SUM(@a BIGINT, @b BIGINT, @c BIGINT) RETURNS @my_sum BIGINT
AS BEGIN
    @temp := @a + @b;
    @my_sum := @temp + @c;
END;
```

Query an SQL function

You can query an SQL function in the same way as querying a Java or Python UDF. Example:

```
DESC FUNCTION my_add;
```

Delete an SQL function

You can delete an SQL function in the same way as deleting a Java or Python UDF. Example:

```
DROP FUNCTION my_add;
```

Call an SQL function

You can call an SQL function in the same way as calling an existing built-in function. Example:

```
SELECT my_sum(col1, col2, col3) from t;
```

Use temporary SQL functions

You can use temporary SQL functions if you no longer need to save SQL functions to the metadata system of MaxCompute. The temporary SQL functions are valid only in the current script. Example:

```
FUNCTION MY_ADD(@a BIGINT) AS @a + 1;
SELECT MY_ADD(key), MY_ADD(value) FROM src;
```

Use function type parameters

You can specify built-in functions, UDFs, or SQL functions in function type parameters when you call SQL functions. Example:

```
FUNCTION ADD(@a BIGINT) AS @a + 1;
FUNCTION OP(@a BIGINT, @fun FUNCTION (BIGINT) RETURNS BIGINT) AS @fun(@a);
SELECT OP(key, ADD), OP(key, ABS) FROM VALUES (1), (2) AS t (key);
```

Returned results:

```
+-----+-----+
| _c0    | _c1    |
+-----+-----+
| 2      | 1      |
| 3      | 2      |
+-----+-----+
```

In this example, two input parameters are specified for the OP function. The @a parameter specifies a value of the BIGINT type. The @fun parameter specifies a function whose input and output parameters are both of the BIGINT type. The OP function transfers @a to the function that is specified by @fun and then to the ADD and ABS functions for processing.

Use anonymous functions

You can specify anonymous functions in function type parameters when you call SQL functions. Example:

```
FUNCTION OP(@a BIGINT, @fun FUNCTION (BIGINT) RETURNS BIGINT) AS @fun(@a);
SELECT OP(key, FUNCTION (@a) AS @a + 1) FROM VALUES (1), (2) AS t (key);
```

In this example, FUNCTION (@a) AS @a + 1 is an anonymous function. You do not need to specify a data type for the input parameter @a. The compiler will infer the data type of @a based on the parameter definition of the OP function.

6.7.9. Embedded UDF

6.7.9.1. Background information

This topic describes the background information of code-embedded user-defined functions (UDFs).

The code-embedded UDFs of MaxCompute resolve the following issues in code implementation and maintenance:

- **Complicated code implementation:** After you create UDFs and develop code, you must compile the code in Java and create resources and functions.
- **Inconvenient code maintenance:** You cannot directly view the implementation logic of the UDFs that are referenced in SQL scripts or obtain the source code of JAR packages.
- **Poor code readability:** To implement Java library functions by using user-defined types (UDTs), you must write Java code as expressions in long code lines. In addition, you may fail to write some Java code as expressions.

Example:

```
Foo f = new Foo();
f.execute();
f.getResult();
```

6.7.9.2. Feature summary

This topic describes the features of code-embedded user-defined functions (UDFs).

Code-embedded UDFs allow you to embed Java or Python code into SQL scripts. When you compile a script, Janino-compiler identifies and extracts the embedded code, compiles the code in Java, and then dynamically generates resources and creates temporary functions.

You can place SQL scripts and third-party code lines in the same source code file. This simplifies the usage of user-defined types (UDTs) or UDFs and facilitates routine development and maintenance.

6.7.9.3. Limits

This topic describes the limits on code-embedded user-defined functions (UDFs).

You can use only Janino-compiler to compile embedded Java code. The syntax of the embedded Java code must be a subset of the standard JDK syntax. Embedded Java code has the following limits:

- Lambda expressions are not supported.
- You cannot specify multiple types of exceptions in a single catch block, for example, `catch(Exception1 | Exception2 e)`.
- Generics cannot be automatically inferred, for example, `Map map = new HashMap<>()`;
- Expressions for type parameter inference are ignored and you must use cast expressions to specify the parameter type, for example, `(String) myMap.get(key)`.
- Assertions are forcibly enabled, even if the `-ea` option of the Java Virtual Machine (JVM) is used.
- Code that is programmed in versions later than Java 8 is not supported.

6.7.9.4. Examples

This topic describes how to use code-embedded UDFs and provides examples for reference.

Reference embedded code in a UDT

Example:

```
SELECT
  s,
  com.mypackage.Foo.extractNumber(s)
FROM VALUES ('abc123def'), ('apple') AS t(s);
#CODE ('lang'='JAVA')
package com.mypackage;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Foo {
    final static Pattern compile = Pattern.compile(".*?([0-9]+).*");
    public static String extractNumber(String input) {
        final Matcher m = compile.matcher(input);
        if (m.find()) {
            return m.group(1);
        }
        return null;
    }
}
#END CODE;
```

Description:

- `#CODE` indicates the beginning of the embedded code block. `#END CODE` indicates the end of the embedded code block. In this example, the embedded code block is placed at the end of the script and applies to the whole script.
- `'lang'='JAVA'` indicates that the embedded code is in Java. `JAVA` can be replaced with `PYTHON` if you compile

code in Python.

- You can use UDT syntax in the SQL script to call `Foo.extractNumber`.

Define and call a Java code-embedded UDF

Example:

```
CREATE TEMPORARY FUNCTION foo AS 'com.mypackage.Reverse' USING
#CODE ('lang'='JAVA')
package com.mypackage;
import com.aliyun.odps.udf.UDF;
public class Reverse extends UDF {
    public String evaluate(String input) {
        if (input == null) return null;
        StringBuilder ret = new StringBuilder();
        for (int i = input.toCharArray().length - 1; i >= 0; i--) {
            ret.append(input.toCharArray()[i]);
        }
        return ret.toString();
    }
}
#END CODE;
SELECT foo('abdc');
```

Description:

- You can place the embedded code block next to `USING` or at the end of the script. If you place it next to `USING`, it applies only to the `CREATE TEMPORARY FUNCTION` statement.
- The function created by `CREATE TEMPORARY FUNCTION` is a temporary function. This temporary function is executed only during the current execution process and is not stored in the MaxCompute meta system.

Define and call a Java code-embedded UDTF

Example:

```
CREATE TEMPORARY FUNCTION foo AS 'com.mypackage.Reverse' USING
#CODE ('lang'='JAVA', 'filename'='embedded.jar')
package com.mypackage;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve({"string->string,string"})
public class Reverse extends UDTF {
    @Override
    public void process(Object[] objects) throws UDFException {
        String str = (String) objects[0];
        String[] split = str.split(",");
        forward(split[0], split[1]);
    }
}
#END CODE;
SELECT foo('ab,dc') AS (a,b);
```

The return value of `@Resolve` must be of the `string[]` type. However, Janino-compiler cannot identify `"string->string,string"` in embedded code as `string[]`. To enable Janino-compiler to identify `"string->string,string"` as `string[]`, `"string->string,string"` must be enclosed in braces `{}`. If you create a Java UDTF by using a common method, braces `{}` are not required.

Define and call a Python code-embedded UDF

Example:

```
CREATE TEMPORARY FUNCTION foo AS 'embedded.UDFTest' USING
#CODE ('lang'='PYTHON', 'filename'='embedded')
from odps.udf import annotate
@annotate("bigint->bigint")
class UDFTest(object):
    def evaluate(self, a):
        return a * a
#END CODE;
SELECT foo(4);
```

Description:

- The indentation of Python code must comply with the specifications of the Python language.
- When you create a Python UDF, the class name that follows the AS clause must contain the file name of the Python source code. You can use `'filename'='embedded'` to specify a virtual file name.

6.8. UDTs

6.8.1. Scenarios and limits

This topic describes the scenarios and limits of user-defined types (UDTs).

MaxCompute introduces UDTs based on the new-generation SQL engine. UDTs allow you to reference classes or objects of third-party programming languages in SQL statements to call methods or obtain data.

UDTs are suitable for the following scenarios:

- You want to use some features that are not provided by MaxCompute but can be implemented in other programming languages.
For example, to implement some features, you need only to call built-in Java classes once. However, MaxCompute does not provide methods to implement these features. If you use user-defined functions (UDFs) to run these tasks, the procedure is complex.
- You want to call a third-party library in SQL statements to implement the related features. In this scenario, UDFs allow you to directly use a function provided by a third-party library in a SQL statement, instead of wrapping the function inside a UDF.
- You want to directly call the source code of a third-party programming language in SQL statements. The SELECT TRANSFORM statement allows you to write scripts to SQL statements. This improves readability and facilitates code maintenance. For some programming languages, such as Java, the source code can be executed only after it is compiled. You can use UDTs to reference objects and classes of these languages in SQL statements.

Limits

- UDTs support only Java. By default, all classes of SDK for Java can be referenced by UDTs.

 **Note** JDK 1.8 is used. A version later than JDK 1.8 may not be supported.

- All operators use the semantics of MaxCompute SQL instead of UDTs.
- UDTs cannot be used as shuffle keys in clauses, such as JOIN, GROUP BY, DISTRIBUTE BY, SORT BY, ORDER BY, or CLUSTER BY.
- DDL statements do not support UDTs. You cannot create tables that contain UDT objects. The final output cannot be UDT types.

6.8.2. Feature summary

This topic describes the features of user-defined types (UDTs) and provides examples for reference.

UDTs supported by many SQL engines are similar to the STRUCT type in MaxCompute. UDTs supported by MaxCompute are similar to the CREATE TYPE statement. A UDT contains both fields and methods. You do not need to use data definition language (DDL) statements to define new data types in MaxCompute. Instead, MaxCompute allows you to reference new data types directly in SQL statements. The following examples show how to use UDTs.

For example, to call the java.lang package in SQL statements, you can use one of the following methods:

- Use UDTs to call the java.lang package

```
-- Enable new data types. A new type of INTEGER (INT) is used in this example.
set odps.sql.type.system.odps2=true;
SELECT java.lang.Integer.MAX_VALUE;
```

Based on Java conventions, you can also omit the java.lang package from the preceding statement and use the following statement:

```
set odps.sql.type.system.odps2=true;
SELECT Integer.MAX_VALUE;
```

Returned results:

```
+-----+
| max_value |
+-----+
| 2147483647 |
+-----+
```

- Use user-defined functions (UDFs) to call the java.lang package

i. Define a UDF class.

```
package com.aliyun.odps.test;
public class IntegerMaxValue extends com.aliyun.odps.udf.UDF {
    public Integer evaluate() {
        return Integer.MAX_VALUE;
    }
}
```

ii. Compile the UDF into a JAR package, upload the package, and create a function.

```
add jar odps-test.jar;
create function integer_max_value as 'com.aliyun.odps.test.IntegerMaxValue' using 'odps-test.jar';
```

iii. Call the function in the SQL statement.

```
select integer_max_value();
```

In this example, UDTs simplify the procedure for you to use other programming languages to extend SQL features.

6.8.3. Implementation principles and feature description

This topic describes the implementation principles of user-defined types (UDTs) and their features.

Implementation principles

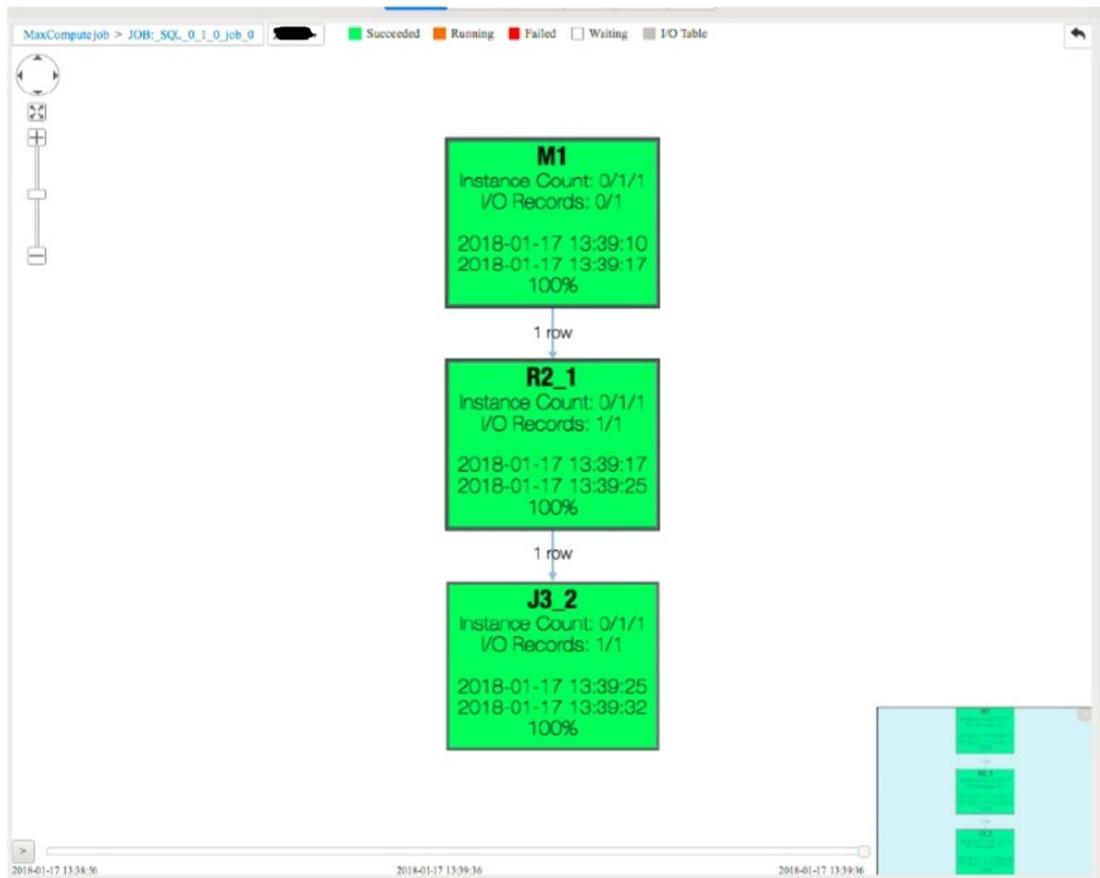
The following example shows how to run a UDT:

```
-- Sample data.
@table1 := select * from values ('10000000000000000000') as t(x);
@table2 := select * from values (100L) as t(y);
-- Code logic.
-- Create an object by using the new method.
@a := select new java.math.BigInteger(x) x from @table1;
-- Call a static method.
@b := select java.math.BigInteger.valueOf(y) y from @table2;
-- Call an instance method.
select /*+mapjoin(b)*/ x.add(y).toString() from @a a join @b b;
```

Returned results:

```
1000000000000000000100
```

The following figure shows the process.



This UDT has three stages: M1, R2, and J3. If a JOIN operation is used in MapReduce, data must be reshuffled. As a result, data is processed at multiple stages. The processes and physical machines that process data vary at different stages.

Only the `new java.math.BigInteger(x)` method is called at the M1 stage.

The `java.math.BigInteger.valueOf(y)` and `x.add(y).toString()` methods are separately called at the J3 stage. These methods are called at different stages and executed in different processes and on different physical machines. UDTs encapsulate these stages to achieve an effect similar to all the stages being implemented on the same Java Virtual Machine (JVM).

The preceding example shows that the result of a subquery supports UDT columns. The `x` column retrieved by variable `a` is of the `java.math.BigInteger` type rather than a built-in type. You can transfer the UDT data to another operator and then call its method. You can also use the UDT data in a data shuffle.

Feature description

- UDTs support only Java. By default, all classes of SDK for Java can be referenced by UDTs.
- UDTs also allow you to upload JAR packages and directly reference these packages. Some flags are provided for UDTs.
 - `set odps.sql.session.resources`: specifies the resource that you want to reference. Separate multiple resources with commas (,). For example, you can set this flag to `foo.sh,bar.txt`.

Note This flag works the same as the flag used to specify resources in the SELECT TRANSFORM statement. Therefore, this flag controls two features.

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.resources=odps-test.jar;
-- Specify the JAR package that you want to reference. You must upload the package to your project in advance.
select new com.aliyun.odps.test.IntegerMaxValue().evaluate();
```

- `odps.sql.session.java.imports`: specifies the default Java package. You can specify multiple packages and separate them with commas (,). This flag is similar to the IMPORT statement in Java. You can specify a class path, such as `java.math.BigInteger`, or use asterisks (*). static import is not supported.

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.resources=odps-test.jar;
set odps.sql.session.java.imports=com.aliyun.odps.test.*;
-- Specify the default JAR package.
select new IntegerMaxValue().evaluate();
```

- UDTs support resource access. In MaxCompute SQL, you can call the static method `com.aliyun.odps.udf.impl.UDTExecutionContext.get()` to obtain the ExecutionContext object. Then, you can use this object to access the current ExecutionContext class and then access resources, such as files or tables.
- UDTs support the following operations:
 - Create objects by using the `new` method.
 - Create arrays by using the `new` method. Initializer lists can be used. Example: `new Integer[] { 1, 2, 3 }`.
 - Call methods, including static methods.

Note

- The identifiers in UDTs contain the names of packages, classes, methods, and fields. All identifiers are case-sensitive.
- Anonymous classes and lambda expressions are not supported.
- UDTs are used in expressions. Functions that do not return values cannot be called in expressions. This issue will be resolved in later versions.

- UDTs support the following data types:

- UDTs support SQL type conversions, such as `cast(1 as java.lang.Object)`. UDTs do not support Java type conversions, such as `(Object)1`.
- The built-in types of UDTs have a one-to-one mapping relationship with specific Java types.
 - You can directly call the method of the Java type to which the built-in type is mapped. Example: `'123'.length()`, `1L.hashCode()`.
 - UDTs can be used in built-in functions and UDFs. For example, in `chr(Long.valueOf('100'))`, `Long.valueOf` returns a value of the `java.lang.Long` type. The built-in function `CHR` supports the built-in `BIGINT` type.
 - The data of a Java primitive type is automatically converted to the boxing type and the preceding two rules apply.

Note For some new built-in data types, you must use `set odps.sql.type.system.odps2=true;` to declare these types. Otherwise, an error occurs.

- The following type conversion rules apply in UDTs:
 - UDT objects can be implicitly converted to the objects of their base classes.
 - UDT objects can be forcibly converted to the objects of their base classes or subclasses.
 - The data type conversion between two objects without inheritance follows the original conversion rules. However, such conversions may result in changes to the data. For example, data of the `java.lang.Long` type can be forcibly converted to the `java.lang.Integer` type. This conversion uses the rules that are used to convert the built-in `BIGINT` type to the `INT` type. This process may result in changes to the data and even loss of data precision.
- UDTs support Java generics. For example, the compiler can determine that the value returned by `java.util.Arrays.asList(new java.math.BigInteger('1'))` is of the `java.util.List<java.math.BigInteger>` type based on the parameter type.

Note You must specify the type parameter in a constructor function or use `java.lang.Object`. This is the same as Java.

- All operators use the semantics of MaxCompute SQL. Examples:
 - Combination of strings: The result of `String.valueOf(1) + String.valueOf(2)` is 3. The two strings are implicitly converted to `DOUBLE`-type values and summed. If you use Java string concatenation to combine the strings, the result is 12.
 - = operations: The `=` operator in SQL statements is used as a comparison operator. It is used to compare one expression with another. You must call the `equals` method in Java to check whether two objects are equivalent. The `=` operator cannot be used to verify the equivalence of two objects.
- UDTs do not have a clear definition of object equality. This is caused by data reshuffling. Objects may be transmitted between different processes or physical machines. During object transmission, an object may be referenced as two different objects. For example, an object may be shuffled to two machines and then reshuffled. Therefore, when you use UDTs, you must use the `equals` method instead of the `=` operator to verify the equivalence of two objects.

Objects in the same row or column are correlated in some way. However, a correlation between objects in different rows or columns cannot be ensured.
- UDTs cannot be used as shuffle keys in clauses, such as `JOIN`, `GROUP BY`, `DISTRIBUTE BY`, `SORT BY`, `ORDER BY`, or `CLUSTER BY`.
- UDTs can be used at the stages of expressions, but cannot be used as outputs.
- You can use UDTs to implement the feature provided by the `SCALAR` function. You can use the built-in functions `COLLECT_SET` and `EXPLODE` with UDTs to implement the features provided by aggregate and table-valued functions.

6.8.4. Benefits

This topic describes the benefits of user-defined types (UDTs).

UDTs deliver the following benefits:

- UDTs are easy to use. You do not need to define functions.
- UDTs support all Java Development Kit (JDK) features. This improves SQL flexibility.
- UDT code can be stored in the same file as the SQL code. This facilitates code management.
- You can directly reference the libraries of other programming languages and reuse code that you have written in other languages.

6.8.5. Performance advantages

This topic describes the performance advantages of user-defined types (UDTs).

UDTs run in a similar way to UDFs in terms of performance. Therefore, the performance of UDTs is almost the same as that of UDFs. The optimized computing engine improves the performance of UDTs in specific scenarios.

- If a UDT object is used in different processes, it must be serialized and deserialized. If you use a UDT to perform operations that do not require data reshuffling, such as JOIN or AGGREGATE, the overheads of serialization and deserialization are avoided.
- The runtime of UDTs is based on Codegen rather than reflection. Therefore, no performance loss occurs. Multiple UDTs can be executed in a single function call. In the following example, it seems that a UDT is called multiple times, but the UDF is actually called only once. Therefore, no additional interface overheads are caused even though the operational units of UDTs are small.

```
values[x].add(values[y]).divide(java.math.BigInteger.valueOf(2))
```

6.8.6. Security advantages

This topic describes the security advantages of user-defined types (UDTs).

Similar to UDFs, UDTs are restricted in the Java sandbox model. To perform restricted operations, you must cancel sandbox isolation for the operations or apply to join a sandbox whitelist.

6.8.7. More examples

6.8.7.1. Example of using Java arrays

This topic provides an example of using Java arrays of user-defined types (UDTs).

Example:

```
set odps.sql.type.system.odps2=true;
set odps.sql.udt.display.toString=true;
SELECT
    new Integer[10],      -- Create an array that contains 10 elements.
    new Integer[] {c1, c2, c3}, -- Initialize an ArrayList to create an array that contains three e
lements.
    new Integer[][] { new Integer[] {c1, c2}, new Integer[] {c3, c4} }, -- Create a multidimensiona
l array.
    new Integer[] {c1, c2, c3} [2], -- Access the elements in the array by using indexes.
    java.util.Arrays.asList(c1, c2, c3); -- Create a list of the List<Integer> type, which can be
used as an array of the Array<Int> type. This is another way to create a built-in array.
FROM VALUES (1,2,3,4) AS t(c1, c2, c3, c4);
```

6.8.7.2. Example of using JSON

This topic provides an example of using JSON for user-defined types (UDTs).

The runtime of a UDT carries a JSON dependency (V2.2.4), which can be directly used in JSON.

Note In addition to JSON dependencies, MaxCompute runtime also carries other dependencies, including commons-logging (1.1.1), commons-lang (2.5), commons-io (2.4), and protobuf-java (2.4.1).

Example:

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*,java.com.google.gson.*; -- To import multiple packages
at a time, separate the packages with commas (,).
@a := select new Gson() gson; -- Create a Gson object.
select
gson.toJson(new ArrayList<Integer>(Arrays.asList(1, 2, 3))), -- Convert an object to a JSON string.
cast(gson.fromJson('["a","b","c"]', List.class) as List<String>) -- Deserialize the JSON string. Gson
forcibly converts the deserialization result from the List<Object> type to the List<String> type.
from @a;
```

Returned results:

```
+-----+-----+
| _c0      | _c1      |
+-----+-----+
| [1,2,3]  | 1        |
+-----+-----+
```

Compared with the built-in function GET_JSON_OBJECT, this UDT-based method is simpler. In addition, this method deserializes the content that is extracted from the JSON string to a supported data type. The deserialization improves efficiency.

6.8.7.3. Example of using complex data types

This topic provides an example of using complex data types of user-defined types (UDTs).

The built-in data type ARRAY maps the java.util.List method, and the built-in data type MAP maps the java.util.Map method.

- Java objects in classes that implement java.util.List or java.util.Map can be used to process complex-type data in MaxCompute SQL.
- MaxCompute can directly call java.util.List or java.util.Map to process data of the ARRAY or MAP type.

Example:

```

set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*;
select
    size(new ArrayList<Integer>()),          -- Call the built-in function size() to obtain the size o
f the ArrayList.
    array(1,2,3).size(),                    -- Call the built-in function size() of the java.util.Lis
t method for the built-in data type ARRAY.
    sort_array(new ArrayList<Integer>()),   -- Sort the data in the ArrayList.
    al[1],                                  -- java.util.List does not support indexing but can proces
s data of the ARRAY type that supports indexing.
    Objects.toString(a),                    -- Convert data from the ARRAY type to the STRING type.
    array(1,2,3).subList(1, 2)              -- Obtain a sublist.
from (select new ArrayList<Integer>(array(1,2,3)) as al, array(1,2,3) as a) t;

```

Returned results:

```

+-----+-----+-----+-----+-----+-----+
| _c0    | _c1    | _c2    | _c3    | _c4    | _c5    |
+-----+-----+-----+-----+-----+-----+
| 0      | 3      | []     | 2      | [1, 2, 3] | [2]    |
+-----+-----+-----+-----+-----+-----+

```

6.8.7.4. Aggregation example

This topic provides an example of using user-defined types (UDTs) to aggregate data.

To aggregate data by using a UDT, you must use the built-in function `COLLECT_SET` or `COLLECT_LIST` to aggregate data to a list and then call the UDT to calculate the aggregate value.

The following example shows how to calculate the median of `BigInteger` data. You cannot directly call the built-in function `MEDIAN` because the data is of the `java.math.BigInteger` type.

```

set odps.sql.session.java.imports=java.math.*;
@test_data := select * from values (1), (2), (3), (5) as t(value);
@a := select collect_list(new BigInteger(value)) values from @test_data; -- Aggregate the data to a
list.
@b := select sort_array(values) as values, values.size() cnt from @a; -- Sort the data.
@c := select if(cnt % 2 == 1, new BigDecimal(values[cnt div 2]), new BigDecimal(values[cnt div 2 - 1]
).add(values[cnt div 2])).divide(new BigDecimal(2)) med from @b;
-- Obtain the final output.
select med.toString() from @c;

```

Returned results:

```

+-----+
| _c0    |
+-----+
| 2.5    |
+-----+

```

The `COLLECT_LIST` function cannot be used to aggregate partial data because it can only aggregate all data in a specific group. It is less efficient than the built-in aggregate functions of MaxCompute or user-defined aggregate functions (UDAFs). We recommend that you use built-in aggregate functions if possible. If all data in a group is aggregated, data skew may occur.

The UDT-based method produces a higher efficiency than UDAFs or built-in aggregate functions, such as WM_CONCAT, for aggregating all data in a group.

6.8.7.5. Example of using table-valued functions

This topic provides an example of table-valued functions of user-defined types (UDTs).

Table-valued functions allow you to specify multiple input rows and columns, and can generate multiple output rows and columns. To implement a table-valued function, perform the following steps:

- Specify multiple input rows or columns. For more information, see [Example of aggregation](#).
- Call the java.util.List or java.util.Map method to generate a data collection, and then call the explode function to split the collection into multiple rows.
- Call different getter methods to obtain data from different fields in a UDT. The obtained data is returned in multiple columns.

The following example shows how to expand a JSON string.

```
@a := select '[{"a": "1", "b": "2"}, {"a": "1", "b": "2"}]' str; -- The sample data.
@b := select new com.google.gson.Gson().fromJson(str, java.util.List.class) l from @a; -- Deserialize the JSON string.
@c := select cast(e as java.util.Map<Object, Object>) m from @b lateral view explode(l) t as e; -- Call the explode function to split the string.
@d := select m.get('a') as a, m.get('b') as b from @c; -- Return the splitting result in multiple columns.
select a.toString() a, b.toString() b from @d; -- The final output. Columns a and b in the variable d are of the Object type.
```

Returned results:

```
+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
+-----+-----+
| 1      | 2      |
+-----+-----+
```

6.9. UDJ

6.9.1. Background information

This topic describes the background information of user-defined join (UDJ).

MaxCompute provides multiple JOIN methods, including INNER JOIN, RIGHT JOIN, OUTER JOIN, LEFT JOIN, FULL JOIN, SEMI JOIN, and ANTI-SEMI JOIN. You can use these built-in JOIN methods in most scenarios. However, these methods are insufficient if you want to perform cross join operations.

In most cases, you can use user-defined functions (UDFs) to describe your code framework. However, the current UDF, user-defined table-valued function (UDTF), and user-defined aggregate function (UDAF) frameworks can only handle one table at a time. To perform user-defined operations for multiple tables, you must use built-in JOIN methods, UDFs, UDTFs, and complex SQL statements. In such scenarios, you must use a custom MapReduce framework instead of SQL to complete the required computing tasks.

Regardless of the scenario, these operations require technological expertise and may cause the following issues:

- In scenarios where you use built-in JOIN methods, UDFs, UDTFs, and complex SQL statements: The use of multiple JOIN methods and code in SQL statements results in a logical black box, which makes it difficult to

generate an optimal execution plan.

- In scenarios where you use a custom MapReduce framework: Execution plans are hard to optimize. Most of the MapReduce code is written in Java. During the deep optimization of native runtime code, the execution of the MapReduce code is less efficient than the execution of the MaxCompute code that is generated by the LLVM code generator.

MaxCompute introduces UDJ to the UDF framework based on the MaxCompute V2.0 computing engine. You can flexibly join tables by using UDJ to perform more customized operations. This simplifies MapReduce-based underlying operations in the distributed system.

6.9.2. Examples of using UDJ

6.9.2.1. Cross join operation by using UDJ

This topic describes how to use user-defined join (UDJ) to perform cross join operations and provides an example for reference.

Assume that two log tables named `payment` and `user_client_log` exist.

- The `payment` table stores the payment records of users. Each payment record contains the user ID, payment time, and payment content. The following table lists the sample data.

user_id	time	pay_info
2656199	2018-02-13 22:30:00	gZhvdYSOQb
8881237	2018-02-13 08:30:00	pYvotULDIT
8881237	2018-02-13 10:32:00	KBuMzRpsko

- The `user_client_log` table stores the client logs of users. Each log contains the user ID, logging time, and log content. The following table lists the sample data.

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzIDyL
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzIDyL

Requirement: For each record in the `user_client_log` table, find the payment record that has the closest time to this record in the `payment` table. Then, join the two records and generate results. The following table lists the results.

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn, pay pYvotULDIT
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzIDyL, pay pYvotULDIT
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzIDyL, pay KBuMzRpsko

To meet this requirement, use one of the following methods:

- Use built-in JOIN methods. SQL sample code:

```

SELECT
  p.user_id,
  p.time,
  merge(p.pay_info, u.content)
FROM
  payment p RIGHT OUTER JOIN user_client_log u
ON p.user_id = u.user_id and abs(p.time - u.time) = min(abs(p.time - u.time))

```

If you join two rows in the tables, you must calculate the minimum difference between the p.time and u.time under the same user_id. However, you cannot call aggregate functions in the join condition. Therefore, you cannot use standard JOIN methods to complete this task.

- Use the UDJ method.
 - i. Create a UDJ function.
 - a. Configure the SDK of the new version.

```

<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.30.0</version>
  <scope>provided</scope>
</dependency>

```

- b. Write UDJ code and package the code as odps-udj-example.jar.

```

package com.aliyun.odps.udf.example.udj;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.Yieldable;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDJ;
import com.aliyun.odps.udf.annotation.Resolve;
import java.util.ArrayList;
import java.util.Iterator;
/** For each record in the table on the right, find the record in the table on the left th
at has the closest time to this record.
 * Join the two records.
 */
@Resolve("->string,bigint,string")
public class PayUserLogMergeJoin extends UDJ {
  private Record outputRecord;
  /** Call the preceding code before data is processed. You can use the PayUserLogMergeJoi
n method for initialization.
  */
  @Override
  public void setup(ExecutionContext executionContext, DataAttributes dataAttributes) {
    //
    outputRecord = new ArrayRecord(new Column[]{
      new Column("user_id", OdpsType.STRING),
      new Column("time", OdpsType.BIGINT),
      new Column("content", OdpsType.STRING)
    });
  }
  /** Rewrite the PayUserLogMergeJoin method to implement the connection logic.
  * @param key: the current key for joining.
  * @param left: the record group of the current key in the table on the left.
  */
}

```

```

    * @param right: the record group of the current key in the table on the right.
    * @param output: generates the result of UDJ.
    */
    @Override
    public void join(Record key, Iterator<Record> left, Iterator<Record> right, Yieldable<Record> output) {
        outputRecord.setString(0, key.getString(0));
        if (! right.hasNext()) {
            // The group on the right is empty. Do not perform operations on this group.
            return;
        } else if (! left.hasNext()) {
            // The group on the left is empty. Generate all records from the group on the right
            but do not join the records of the two tables.
            while (right.hasNext()) {
                Record logRecord = right.next();
                outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
                outputRecord.setString(2, logRecord.getString(1));
                output.yield(outputRecord);
            }
            return;
        }
        ArrayList<Record> pays = new ArrayList<>();
        // The records in the group on the left are iterated from the beginning to the end.
        // The iterator cannot reset the records in the group on the right.
        // Save each record of the group on the left to ArrayList.
        left.forEachRemaining(pay -> pays.add(pay.clone()));
        while (right.hasNext()) {
            Record log = right.next();
            long logTime = log.getDatetime(0).getTime();
            long minDelta = Long.MAX_VALUE;
            Record nearestPay = null;
            // Iterate all the records of the group on the left. Then, you can find the minimum
            time difference between the records.
            for (Record pay: pays) {
                long delta = Math.abs(logTime - pay.getDatetime(0).getTime());
                if (delta < minDelta) {
                    minDelta = delta;
                    nearestPay = pay;
                }
            }
            // Merge the log records and payment records that are closest in time and then generate
            results.
            outputRecord.setBigint(1, log.getDatetime(0).getTime());
            outputRecord.setString(2, mergeLog(nearestPay.getString(1), log.getString(1)));
            output.yield(outputRecord);
        }
    }
    String mergeLog(String payInfo, String logContent) {
        return logContent + ", pay " + payInfo;
    }
    @Override
    public void close() {
    }
}

```

c. Add the odps-udj-example.jar package to MaxCompute.

```
add jar odps-udj-example.jar;
```

d. Register the UDJ function `pay_user_log_merge_join` in MaxCompute.

```
create function pay_user_log_merge_join
  as 'com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin'
  using 'odps-udj-example.jar';
```

ii. Prepare sample data.

a. Create the payment table and the `user_client_log` table.

```
create table payment(user_id string,time datetime,pay_info string);
create table user_client_log(user_id string,time datetime,content string);
```

b. Insert data into the sample tables.

```
-- Insert data into the payment table.
INSERT OVERWRITE TABLE payment VALUES
('1335656', datetime '2018-02-13 19:54:00', 'PEqMSHyktn'),
('2656199', datetime '2018-02-13 12:21:00', 'pYvotuLDIT'),
('2656199', datetime '2018-02-13 20:50:00', 'PEqMSHyktn'),
('2656199', datetime '2018-02-13 22:30:00', 'gZhvdysOQb'),
('8881237', datetime '2018-02-13 08:30:00', 'pYvotuLDIT'),
('8881237', datetime '2018-02-13 10:32:00', 'KBuMzRpsko'),
('9890100', datetime '2018-02-13 16:01:00', 'gZhvdysOQb'),
('9890100', datetime '2018-02-13 16:26:00', 'MxONdLckwa')
;

-- Insert data into the user_client_log table.
INSERT OVERWRITE TABLE user_client_log VALUES
('1000235', datetime '2018-02-13 00:25:36', 'click FNOXAibRjkIaQPB'),
('1000235', datetime '2018-02-13 22:30:00', 'click GczrYaxvkiPultZ'),
('1335656', datetime '2018-02-13 18:30:00', 'click MxONdLckpAFUHRs'),
('1335656', datetime '2018-02-13 19:54:00', 'click mKRPgOciFDyzTgM'),
('2656199', datetime '2018-02-13 08:30:00', 'click CZwafHsbJOPNitL'),
('2656199', datetime '2018-02-13 09:14:00', 'click nYHJqIpjevKkToy'),
('2656199', datetime '2018-02-13 21:05:00', 'click gbAfPCwrGXveJpI'),
('2656199', datetime '2018-02-13 21:08:00', 'click dhpZyWMuGjBOTJP'),
('2656199', datetime '2018-02-13 22:29:00', 'click bAsxnUdDhvfqaBr'),
('2656199', datetime '2018-02-13 22:30:00', 'click XIhZdLaOocQRmry'),
('4356142', datetime '2018-02-13 18:30:00', 'click DYqShmGbIOWKier'),
('4356142', datetime '2018-02-13 19:54:00', 'click DYqShmGbIOWKier'),
('8881237', datetime '2018-02-13 00:30:00', 'click MpkvilgWSmhUuPn'),
('8881237', datetime '2018-02-13 06:14:00', 'click OkTYNUHMqZz1DyL'),
('8881237', datetime '2018-02-13 10:30:00', 'click OkTYNUHMqZz1DyL'),
('9890100', datetime '2018-02-13 16:01:00', 'click vOTQfBFjcgXisYU'),
('9890100', datetime '2018-02-13 16:20:00', 'click WxaLgOCCvevhiFJ')
;
```

iii. Use a UDJ function in SQL.

```
SELECT r.user_id, from_unixtime(time/1000) as time, content FROM (
SELECT user_id, time as time, pay_info FROM payment
) p JOIN (
SELECT user_id, time as time, content FROM user_client_log
) u
ON p.user_id = u.user_id
USING pay_user_log_merge_join(p.time, p.pay_info, u.time, u.content)
r
AS (user_id, time, content);
```

Parameters in the USING clause:

- `pay_user_log_merge_join`: the name of the UDJ function in SQL.
- `(p.time, p.pay_info, u.time, u.content)`: the columns in both of the tables used in the UDJ function.
- `r`: the alias of the result returned by the UDJ function. You can reference this alias in other SQL statements.
- `(user_id, time, content)` are the columns returned by the UDJ function.

Returned results:

```

+-----+-----+-----+
| user_id | time          | content |
+-----+-----+-----+
| 1000235 | 2018-02-13 00:25:36 | click FNOXAibRjkIaQPB |
| 1000235 | 2018-02-13 22:30:00 | click GczrYaxvkiPultZ |
| 1335656 | 2018-02-13 18:30:00 | click MxONdLckpAFUHRs, pay PEqMSHyktn |
| 1335656 | 2018-02-13 19:54:00 | click mKRPGociFDyzTgM, pay PEqMSHyktn |
| 2656199 | 2018-02-13 08:30:00 | click CZwafHsbJOPNitL, pay pYvotuLDIT |
| 2656199 | 2018-02-13 09:14:00 | click nYHJqIpjevKkToy, pay pYvotuLDIT |
| 2656199 | 2018-02-13 21:05:00 | click gbAfPCwrGXvEjpl, pay PEqMSHyktn |
| 2656199 | 2018-02-13 21:08:00 | click dhpZyWMuGjBOTJP, pay PEqMSHyktn |
| 2656199 | 2018-02-13 22:29:00 | click bAsxnUdDhvfqaBr, pay gZhvdySOqb |
| 2656199 | 2018-02-13 22:30:00 | click XIhZdLaOocQRmrY, pay gZhvdySOqb |
| 4356142 | 2018-02-13 18:30:00 | click DYqShmGbIoWKier |
| 4356142 | 2018-02-13 19:54:00 | click DYqShmGbIoWKier |
| 8881237 | 2018-02-13 00:30:00 | click MpkvilgWSmhUuPn, pay pYvotuLDIT |
| 8881237 | 2018-02-13 06:14:00 | click OkTYNUHMqZzldyL, pay pYvotuLDIT |
| 8881237 | 2018-02-13 10:30:00 | click OkTYNUHMqZzldyL, pay KBuMzRpsko |
| 9890100 | 2018-02-13 16:01:00 | click vOTQfBFjcgXisYU, pay gZhvdySOqb |
| 9890100 | 2018-02-13 16:20:00 | click WxaLgOCcVEvhiFJ, pay MxONdLckwa |
+-----+-----+-----+

```

6.9.2.2. Pre-sorting

This topic describes the pre-sorting feature of user-defined join (UDJ) and provides an example for reference.

An iterator is used to iterate all records in the payment table and find the payment record that has the closest time to a specific log record in the `user_client_log` table. To perform this task, you must load all payment records with the same `user_id` to an `ArrayList`. This method can be applied when the number of payment records is small. Due to the limits of the memory size, you must find another method to load the data if a large number of payment records have been generated. This topic describes how to address this issue by using the `SORT BY` clause.

If the number of payment records is too large to be stored in the memory and all the data in the table has been sorted by time, you need only to compare the first element in the two lists.

This method uses the `SORT BY` clause to pre-sort the UDJ data. To achieve the same effect by using the previous method, you need only to cache a maximum of three data records during pre-sorting. The following code shows an example:

```

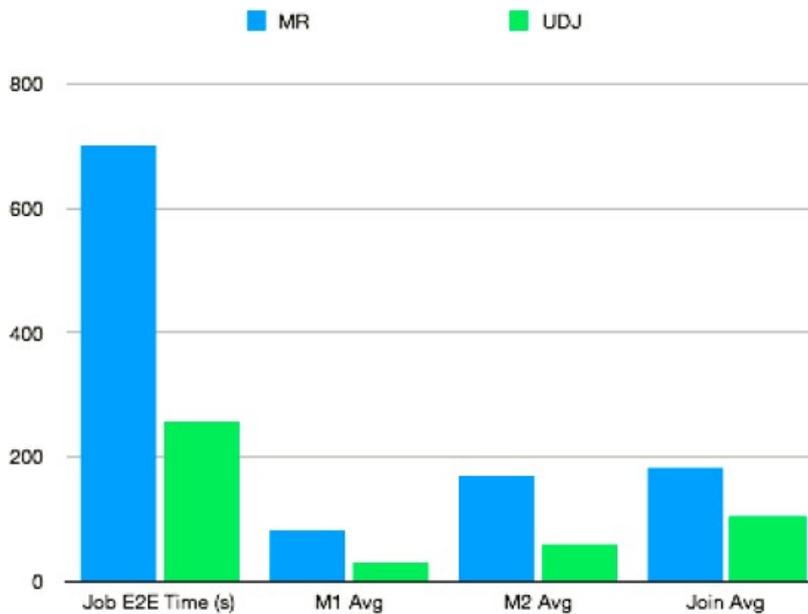
@Override
public void join(Record key, Iterator<Record> left, Iterator<Record> right, Yieldable<Record> output
) {
    outputRecord.setString(0, key.getString(0));
    if (! right.hasNext()) {
        return;
    } else if (! left.hasNext()) {
        while (right.hasNext()) {
            Record logRecord = right.next();
            outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
            outputRecord.setString(2, logRecord.getString(1));
            output.yield(outputRecord);
        }
        return;
    }
    long prevDelta = Long.MAX_VALUE;
    Record logRecord = right.next();
    Record payRecord = left.next();
    Record lastPayRecord = payRecord.clone();
    while (true) {
        long delta = logRecord.getDatetime(0).getTime() - payRecord.getDatetime(0).getTime();
        if (left.hasNext() && delta > 0) {
            // The time delta between the two records decreases and the operation can continue.
            // Explore the group on the left to try to obtain a smaller time delta.
            lastPayRecord = payRecord.clone();
            prevDelta = delta;
            payRecord = left.next();
        } else {
            // The minimum time delta point is reached. Check the final record in the payment table.
            // Generate the merged result and prepare to process the next record.
            // The group on the right.
            Record nearestPay = Math.abs(delta) < prevDelta ? payRecord : lastPayRecord;
            outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
            String mergedString = mergeLog(nearestPay.getString(1), logRecord.getString(1));
            outputRecord.setString(2, mergedString);
            output.yield(outputRecord);
            if (right.hasNext()) {
                logRecord = right.next();
                prevDelta = Math.abs(
                    logRecord.getDatetime(0).getTime() - lastPayRecord.getDatetime(0).getTime()
                );
            } else {
                break;
            }
        }
    }
}

```

6.9.3. Performance

This topic describes the performance of user-defined join (UDJ).

A real online MapReduce job is used as an example to verify the performance of UDJ. The job runs based on a complex algorithm. In this example, two tables are joined, UDJ is used to rewrite the MapReduce job, and the correctness of the UDJ results is checked. The following figure shows the MapReduce and UDJ performance under the same data concurrency.



As shown in the figure, UDJ conveniently describes the complex logic of how to handle multiple tables and greatly improves performance. The code is called only within UDJ. The entire Mapper logic in this example is executed by the native runtime engine of MaxCompute. The data exchange logic between the MaxCompute UDJ runtime engine and Java interfaces is optimized in Java code. The JOIN logic of UDJ is more efficient than that of a reducer.

6.10. Parameterized view

This topic describes the parameterized view feature that is supported by the MaxCompute SQL engine.

In the traditional views of MaxCompute, complex SQL scripts are encapsulated at the underlying layer. Callers can call views the same way as reading a common table without the need to understand the underlying implementation. Traditional views are widely used because they implement encapsulation and reuse. However, traditional views cannot accept parameters from callers. This reduces code reuse efficiency. For example, a caller cannot filter data in the underlying table that is read by a view nor pass other parameters. The MaxCompute SQL engine supports parameterized views and allows you to import tables or variables to customize views.

Define a parameterized view

You can use the following syntax to create a parameterized view:

```
-- view with parameters
-- param @a -a table parameter
-- param @b -a string parameter
-- returns a table with schema (key STRING,value STRING)
CREATE VIEW IF NOT EXISTS pv1(@a table (k STRING,v BIGINT), @b STRING)
AS
SELECT srcp.key,srcp.value FROM srcp JOIN @a ON srcp.key=a.k AND srcp.p=@b;
```

Syntax description:

- To create a parameterized view, you must specify parameters. Therefore, you must use Script Mode SQL to create a parameterized view.
- The created view pv1 has two parameters: TABLE and STRING. The parameter value can be a table or of a basic data type.
- The parameter value can also be a subquery, for example, `SELECT * FROM view_name((SELECT 1 FROM src WH`

```
ERE a > 0), 1); .
```

- When you create a view, you can set ANY for a parameter, which indicates any data type. For example, `CREATE VIEW paramed_view (@a ANY) AS SELECT * FROM src WHERE CASE WHEN @a IS NULL THEN key1 ELSE key2 END = key3;` defines that the first parameter of the view can be a value of any data type.

However, the ANY type cannot be used in a specific operation, such as + or AND, which requires specific data types. A field of the ANY type is often used as a PassThrough column in the TABLE parameter. Example:

```
CREATE VIEW paramed_view (@a TABLE(name STRING, id ANY, age BIGINT)) AS SELECT * FROM @a WHERE name = 'foo' AND age < 25;
-- The call example.
SELECT * FROM param_view((SELECT name, id, age FROM students));
```

Note

- After you execute the `CREATE VIEW` statement to create a view, you can run the `DESC` command to obtain the description of the view. This description contains the return type of the view.
- The return type of a view is recalculated when the view is called. It may be different from the return type, such as ANY, that you specify when you create the view.

- When you create a view, you can use an asterisk (*) in the TABLE parameter to retrieve any columns. The asterisk (*) can represent a specific data type or the ANY type. Example:

```
CREATE VIEW paramed_view (@a TABLE(key STRING, * ANY), @b TABLE(key STRING, * STRING)) AS SELECT a.* FROM @a JOIN @b ON a.key = b.key;
-- The call example.
SELECT name, address FROM param_view((SELECT school, name, age, address FROM student), school) WHERE age < 20;
```

In this example, the view accepts two TABLE parameters. In the table that is specified by the first TABLE parameter, data in the first column is of the STRING type, and data in other columns can be of the ANY type. In the table that is specified by the second TABLE parameter, data in the first and other columns is all of the STRING type. Take note of the following points:

- The varied-length part must be placed at the end of the table that is specified by the TABLE parameter. This means that the * column cannot be followed by other columns. Therefore, the table that is specified by the TABLE parameter can contain only one varied-length column.
- The varied-length part must be placed at the end of the table that is specified by the TABLE parameter. However, the columns of an input table may not be arranged in this sequence. In this case, the columns need to be rearranged. A subquery can be used as a parameter value and must be enclosed in a pair of parentheses ().
- No name is specified for the varied-length part of the table that is specified by the TABLE parameter. As a result, data in the varied-length part cannot be referenced or used for operation when you define the view.
- Although you cannot use the varied-length part for operation, you can use the `SELECT *` statement to transfer data in the varied-length part out of the table.
- The column of the table that is specified by the TABLE parameter may be different from the fixed-length column that you specify when you define the view. If the names are different, the compiler automatically renames the column of the table that is specified by the TABLE parameter. If the data types are different, the compiler performs an implicit conversion. If the implicit conversion fails, an error occurs.

Call a parameterized view

You can execute the following statement to call the pv1 view that you defined:

```

@a := SELECT * FROM src WHERE value >0;
--call view with table variable and scalar
@b := SELECT * FROM pv1(@a,'20170101');
@another_day := '20170102';
--call view with table name and scalar variable
@c := SELECT * FROM pv1(src2, @another_day);
@d := SELECT * FROM @c UNION ALL SELECT * FROM @b;
WITH
t AS(SELECT * FROM src3)
SELECT * FROM @c
UNION ALL
SELECT * FROM @d
UNION ALL
SELECT* FROM pv1(t,@another_day);

```

Note You can use different parameters to call the pv1 view.

- The value of the TABLE parameter can be the name of a physical table, view, or table variable. You can also set the TABLE parameter to a table alias by using common table expressions (CTEs).
- The values of common parameters can be variables or constants.

Usage notes

- The script of a parameterized view can contain only data manipulation language (DML) statements. The statements, such as INSERT, CREATE TABLE, or PRINT, cannot be contained.
- A parameterized view can contain multiple SQL statements.

```

-- view with parameters
-- param @a -a table parameter
-- param @b -a string parameter
-- returns a table with schema (key string,value string)
CREATE VIEW IF NOT EXISTS pv2(@a TABLE (k STRING,v BIGINT), @b STRING) AS
BEGIN
@srcp := SELECT * FROM srcp WHERE p=@b;
@pv2 := SELECT srcp.key,srcp.value FROM @srcp JOIN @a ON srcp.key=a.k;
END;

```

Note Statements between BEGIN and END are the script of this view. The `@pv2 :=...` statement is similar to the RETURN statement in other programming languages. You can use the `@pv2 :=...` statement to assign a value to a variable of an implicit table that has the same name as the view.

- The matching rules for actual and formal view parameters are the same as those specified in a common weakly-typed language. Specifically, if a view parameter can be implicitly converted, it can match the defined parameter. For example, a value of the BIGINT type can match parameters of the DOUBLE type. For table variables, if the schema of table a can be inserted into table b, table a can be used to match the table-type parameters that have the same schema as table b.
- You can explicitly declare the return type to make the code easier to read.

```
CREATE VIEW IF NOT EXISTS pv3(@a table (k STRING,v BIGINT), @b STRING)
RETURNS @ret TABLE (x STRING,y STRING)
AS
BEGIN
    @srcp := SELECT * FROM srcp WHERE p=@b;
    @ret := SELECT srcp.key,srcp.value FROM @srcp JOIN @a ON srcp.key=a.k;
end;
```

- Note** RETURNS @ret TABLE (x STRING,y STRING) defines the following information:
- The return type, which indicates the type returned to the caller. The return type is specified by the TABLE (x STRING,y STRING) parameter. You can use this parameter to customize the table schema.
 - The response parameter. The @ret parameter defines the name of the response parameter. Assigning a value to the response parameter is performed in the view script.

You can consider the view that contains no BEGIN/END or return variables as a simplified parameterized view.

6.11. CLONE TABLE

This topic describes how to use the CLONE TABLE statement. The CLONE TABLE statement is used to clone data from one table to another. This statement improves data migration efficiency.

Limits

- The schema of the destination table must be compatible with that of the source table.
- The CLONE TABLE statement can be executed for partitioned tables and non-partitioned tables. This statement cannot be executed for hash clustering tables.
- If a destination table is created before the CLONE TABLE statement is executed, data in a maximum of 10,000 partitions can be cloned at a time.
- If a destination table is not created before the CLONE TABLE statement is executed, the number of partitions from which you can clone data at a time is unlimited. This ensures atomicity.
- You can execute the CLONE TABLE statement for up to seven times in the same non-partitioned table or in the same partition of a partitioned table.
- You cannot execute the CLONE TABLE statement for projects across regions.

Syntax

```
CLONE TABLE <[src_project_name.]src_table_name> [PARTITION(spec), ...]
TO <[dest_project_name.]desc_table_name> [IF EXISTS (OVERWRITE | IGNORE)] ;
```

Description

The CLONE TABLE statement is used to clone data from the src_table_name table to the desc_table_name table.

- Note** After you clone data to the desc_table_name table, we recommend that you verify the data accuracy. For example, you can execute the SELECT COUNT statement to view the number of rows in the destination table or execute the DESC statement to view the table size.

Parameters

- src_table_name: the name of the source table.
- src_project_name: the name of the project to which the source table belongs. If this parameter is not specified, the name of the current project is used by default.

- desc_table_name: the name of the destination table.
 - If a destination table is not created, the table is created by using the CREATE TABLE LIKE statement when you execute the CLONE TABLE statement.
 - If a destination table is created and IF EXISTS OVERWRITE is specified, data in the partitions of the destination table is overwritten when you execute the CLONE TABLE statement.
 - If a destination table is created and IF EXISTS IGNORE is specified, the existing partitions in the table are skipped and the data in these partitions is not overwritten when you execute the CLONE TABLE statement.
- dest_project_name: the name of the project to which the destination table belongs. If this parameter is not specified, the current project name is used by default.

Examples

Assume that the partitioned table srcpart_copy and non-partitioned table src_copy are source tables. The two tables have the following metadata:

```
odps@ multi>READ srcpart_copy;
+-----+-----+-----+-----+
| key      | value    | ds        | hr      |
+-----+-----+-----+-----+
| 1        | ok49     | 2008-04-09 | 11     |
| 1        | ok48     | 2008-04-08 | 12     |
+-----+-----+-----+-----+
odps@ multi>READ src_copy;
+-----+-----+
| key      | value    |
+-----+-----+
| 1        | ok       |
+-----+-----+
```

- Clone all data from the non-partitioned table src_copy to the destination table src_clone.

```
CLONE TABLE src_copy TO src_clone;
```

Returned results:

```
ID = 2019102303024544g2540cdv2
OK
```

//After the data is cloned, query data in the destination table src_clone and check the data accuracy.

```
SELECT * FROM src_clone;
```

- Clone data from a specified partition of the partitioned table srcpart_copy to the destination table srcpart_clone.

```
CLONE TABLE srcpart_copy PARTITION(ds="2008-04-09", hr='11') TO srcpart_clone IF EXISTS OVERWRITE;
```

Returned results:

```
ID = 20191023030534986g4540cdv2
OK
```

//After the data is cloned, query data in the destination table srcpart_clone and check the data accuracy.

```
SELECT * FROM srcpart_clone;
```

- Clone all data from the partitioned table srcpart_copy to the destination table srcpart_clone and skip the data in the existing partitions of the destination table.

```
CLONE TABLE srcpart_copy TO srcpart_clone IF EXISTS IGNORE;
```

Returned results:

```
ID = 20191023030619196g5540cdv2
OK
```

//After the data is cloned, query data in the destination table srcpart_clone and check the data accuracy.

```
SELECT * FROM srcpart_clone;
```

- Clone all data from the partitioned table srcpart_copy to the destination table srcpart_clone2.

```
CLONE TABLE srcpart_copy TO srcpart_clone2;
```

Returned results:

```
ID = 20191023030825186g6540cdv2
OK
```

//After the data is cloned, query data in the destination table srcpart_clone2 and check the data accuracy.

```
SELECT * FROM srcpart_clone2;
```

6.12. Geographic functions

6.12.1. Usage notes

Before you use geographic functions, understand the following points:

All functions are published in the geospatial project of the DataWorks marketplace. These functions are prefixed with `ST_`. You can click a function to view and use it without the need to apply for permissions. To use a function, add a `geospatial.` project prefix to the beginning of the function name and commit SQL statements that contain this function with the following flags:

```
set odps.sql.hive.compatible=true;
set odps.sql.udf.java.retain.legacy=false;
set odps.isolation.session.enable=true;
```

6.12.2. Constructors

6.12.2.1. ST_AsBinary

This topic describes the Constructor function `ST_AsBinary` and provides an example for reference.

Function declaration:

```
ST_AsBinary(ST_Geometry)
```

Description: This function returns the well-known binary (WKB) expression of the input geometry.

Example:

```
SELECT ST_AsBinary(ST_Point(1, 2)) FROM onerow;
```

Returned results:

```
WKB representation of POINT (1 2)
```

6.12.2.2. ST_AsGeoJson

This topic describes the Constructor function ST_AsGeoJson and provides an example for reference.

Function declaration:

```
ST_AsGeoJson(geometry)
```

Description: This function returns the GeoJSON expression of the input geometry.

Example:

```
SELECT ST_AsGeoJson(ST_Point(1.0, 2.0)) from onerow;
```

Returned results:

```
{"type":"Point", "coordinates":[1.0, 2.0]}
```

6.12.2.3. ST_AsJson

This topic describes the Constructor function ST_AsJson and provides examples for reference.

Function declaration:

```
ST_AsJSON(ST_Geometry)
```

Description: This function returns the JSON expression of the input geometry.

Examples:

```
SELECT ST_AsJSON(ST_Point(1.0, 2.0)) from onerow;
```

Returned results:

```
{"x":1.0, "y":2.0}
```

```
SELECT ST_AsJSON(ST_SetSRID(ST_Point(1, 1), 4326)) from onerow;
```

Returned results:

```
{"x":1.0, "y":1.0, "spatialReference":{"wkid":4326}}
```

6.12.2.4. ST_AsShape

This topic describes the Constructor function ST_AsShape and provides an example for reference.

Function declaration:

```
ST_AsShape(ST_Geometry)
```

Description: This function returns the ESRI shape expression of the input geometry.

Example:

```
SELECT ST_AsShape(ST_Point(1, 2)) FROM onerow;
```

Returned results:

```
Esri shape representation of POINT (1 2)
```

6.12.2.5. ST_AsText

This topic describes the Constructor function `ST_AsText` and provides an example for reference.

Function declaration:

```
ST_AsText(ST_Geometry)
```

Description: This function returns the well-known text (WKT) expression of the input geometry.

Example:

```
SELECT ST_AsText(ST_Point(1, 2)) FROM onerow;
```

Returned results:

```
POINT (1 2)
```

6.12.2.6. ST_GeomCollection

This topic describes the `ST_GeomCollection` function of the Constructors function and provides an example for reference.

Function declaration:

```
ST_GeomCollection(wkt)
```

Description: This function constructs a multi-part geometry based on the well-known text (WKT) representation defined by the Open Geospatial Consortium (OGC).



Notice The `ST_GeomCollection` function in MaxCompute supports only the multi-part geometry feature, not the collection feature.

Example:

```
SELECT ST_GeomCollection('multipoint ((1 0), (2 3))') FROM src LIMIT 1;
-- Construct a multipoint geometry.
ST_GeomCollection('POINT(1 1), LINESTRING(2 0,3 0)')
-- The collection feature is not supported.
```

6.12.2.7. ST_GeomFromGeojson

This topic describes the Constructor function `ST_GeomFromGeojson` and provides an example for reference.

Function declaration:

```
ST_GeomFromGeoJson(json)
```

Description: This function constructs a geometry based on the input GeoJSON expression.

Example:

```
SELECT ST_GeomFromGeoJson('{"type":"Point", "coordinates":[1.2, 2.4]}') FROM src LIMIT 1;
-- Construct a point.
SELECT ST_GeomFromGeoJson('{"type":"LineString", "coordinates":[[1,2], [3,4]]}') FROM src LIMIT 1;
-- Construct a linestring.
```

6.12.2.8. ST_GeomFromJSON

This topic describes the Construct or function ST_GeomFromJSON and provides an example for reference.

Function declaration:

```
ST_GeomFromJSON(json)
```

Description: This function constructs a geometry based on the input ESRI JSON expression.

Example:

```
SELECT ST_GeomFromJSON('{"x":0.0,"y":0.0}') FROM src LIMIT 1;
-- Construct a point.
```

6.12.2.9. ST_GeomFromShape

This topic describes the Construct or function ST_GeomFromShape and provides an example for reference.

Function declaration:

```
ST_GeomFromShape(shape)
```

Description: This function constructs a geometry based on the input ESRI shape expression.

Example:

```
SELECT ST_GeomFromShape(ST_AsShape(ST_Point(1, 2)));
-- Construct a point.
```

6.12.2.10. ST_GeomFromText

This topic describes the Construct or function ST_GeomFromText and provides an example for reference.

Function declaration:

```
ST_GeomFromText(wkt)
```

Description: This function constructs a geometry based on the input well-known text (WKT) representation defined by the Open Geospatial Consortium (OGC).

Example:

```
SELECT ST_GeomFromText('linestring (1 0, 2 3)') FROM src LIMIT 1;
-- Construct a linestring.
SELECT ST_GeomFromText('multipoint ((1 0), (2 3))') FROM src LIMIT 1;
-- Construct a multipoint geometry.
```

6.12.2.11. ST_GeomFromWKB

This topic describes the Construct or function ST_GeomFromWKB and provides an example for reference.

Function declaration:

```
ST_GeomFromWKB(wkb)
```

Description: This function constructs a geometry based on the input well-known binary (WKB) expression that complies with the Open Geospatial Consortium (OGC) standards.

Example:

```
SELECT ST_GeomFromWKB(ST_AsBinary(ST_GeomFromText('linestring (1 0, 2 3)'))) FROM src LIMIT 1;
-- Construct a linestring.
SELECT ST_GeomFromWKB(ST_AsBinary(ST_GeomFromText('multipoint ((1 0), (2 3))'))) FROM src LIMIT 1;
-- Construct a multipoint geometry.
```

6.12.2.12. ST_GeometryType

This topic describes the Construct or function ST_GeometryType and provides examples for reference.

Function declaration:

```
ST_GeometryType(geometry)
```

Description: This function returns the type of the input geometry.

Examples:

```
SELECT ST_GeometryType(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
ST_Point
```

```
SELECT ST_GeometryType(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
ST_LineString
```

```
SELECT ST_GeometryType(ST_Polygon(2,0, 2,3, 3,0)) FROM src LIMIT 1;
```

Returned results:

```
ST_Polygon
```

6.12.2.13. ST_LineString

This topic describes the Construct or function ST_LineString and provides an example for reference.

Function declaration:

```
ST_LineString(x, y, [x, y]*)
ST_LineString('linestring( ... )')
ST_LineString(array(x+), array(y+))
ST_LineString(array(ST_Point(x,y)+))
```

Description: This function constructs a two-dimensional line.

Example:

```
SELECT ST_LineString(1, 1, 2, 2, 3, 3) from src LIMIT 1;
SELECT ST_LineString('linestring(1 1, 2 2, 3 3)') from src LIMIT 1;
SELECT ST_LineString(array(1,2,3), array (1,2,3)) from src LIMIT 1;
SELECT ST_LineString(array(ST_Point(1, 1), ST_Point(2,2), ST_Point(3,3))) from src LIMIT 1;
```

6.12.2.14. ST_LineFromWKB

This topic describes the Construct or function ST_LineFromWKB and provides an example for reference.

Function declaration:

```
ST_LineFromWKB(wkb)
```

Description: This function constructs a two-dimensional line based on the input well-known binary (WKB) representation defined by the Open Geospatial Consortium (OGC).

Example:

```
SELECT ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('linestring (1 0, 2 3)'))) FROM src LIMIT 1;
-- Construct a two-dimensional line.
```

6.12.2.15. ST_MultiLineString

This topic describes the Construct or function ST_MultiLineString and provides an example for reference.

Function declaration:

```
ST_MultiLineString(array(x1, y1, x2, y2, ... ), array(x1, y1, x2, y2, ... ), ... )
ST_MultiLineString('multilinestring( ... )')
```

Description: This function constructs a two-dimensional multilinestring.

Example:

```
SELECT ST_MultiLineString(array(1, 1, 2, 2), array(10, 10, 20, 20)) from src LIMIT 1;
SELECT ST_MultiLineString('multilinestring ((1 1, 2 2), (10 10, 20 20))', 0) from src LIMIT 1;
-- Construct a two-dimensional multilinestring.
```

6.12.2.16. ST_MLineFromWKB

This topic describes the Construct or function `ST_MLineFromWKB` and provides an example for reference.

Function declaration:

```
ST_MLineFromWKB(wkb)
```

Description: This function constructs a two-dimensional multilinestring based on the input well-known binary (WKB) expression that complies with the Open Geospatial Consortium (OGC) standards.

Example:

```
SELECT ST_MLineFromWKB(ST_AsBinary(ST_GeomFromText('multilinestring ((1 0, 2 3), (5 7, 7 5))'))) FROM src LIMIT 1;
-- Construct a two-dimensional multilinestring.
```

6.12.2.17. ST_MultiPoint

This topic describes the Construct or function `ST_MultiPoint` and provides an example for reference.

Function declaration:

```
ST_MultiPoint(x1, y1, x2, y2, x3, y3)
ST_MultiPoint('multipoint( ... )')
```

Description: This function constructs a two-dimensional multipoint geometry.

Example:

```
SELECT ST_MultiPoint(1, 1, 2, 2, 3, 3) from src LIMIT 1;
-- Construct a three-point geometry.
SELECT ST_MultiPoint('MULTIPOINT ((10 40), (40 30))') from src LIMIT 1;
-- Construct a two-point geometry.
```

6.12.2.18. ST_MPointFromWKB

This topic describes the Construct or function `ST_MPointFromWKB` and provides an example for reference.

Function declaration:

```
ST_MPointFromWKB(wkb)
```

Description: This function constructs a two-dimensional multipoint geometry based on the input well-known binary (WKB) representation defined by the Open Geospatial Consortium (OGC).

Example:

```
SELECT ST_MPointFromWKB(ST_AsBinary(ST_GeomFromText('multipoint ((1 0), (2 3))'))) FROM src LIMIT 1;
-- Construct a two-dimensional multipoint geometry.
```

6.12.2.19. ST_MultiPolygon

This topic describes the Construct or function `ST_MultiPolygon` and provides an example for reference.

Function declaration:

```
ST_MultiPolygon(array(x1, y1, x2, y2, ... ), array(x1, y1, x2, y2, ... ), ... )
ST_MultiPolygon('multipolygon ( ... )')
```

Description: This function constructs a two-dimensional multipolygon.

Example:

```
SELECT ST_MultiPolygon(array(1, 1, 1, 2, 2, 2, 2, 1), array(3, 3, 3, 4, 4, 4, 4, 3)) from src LIMIT 1;
SELECT ST_MultiPolygon('multipolygon (((0 0, 0 1, 1 0, 0 0)), ((2 2, 2 3, 3 2, 2 2)))') from src LIMIT 1;
-- Construct a two-dimensional multipolygon.
```

6.12.2.20. ST_MPolyFromWKB

This topic describes the Construct or function ST_MPolyFromWKB and provides an example for reference.

Function declaration:

```
ST_MPolyFromWKB(wkb)
```

Description: This function constructs a two-dimensional multipolygon based on the input well-known binary (WKB) expression that complies with the Open Geospatial Consortium (OGC) standards.

Example:

```
SELECT ST_MPolyFromWKB(ST_AsBinary(ST_GeomFromText('multipolygon (((0 0, 1 0, 0 1, 0 0)), ((2 2, 1 2, 2 1, 2 2)))')) FROM src LIMIT 1;
-- Construct a two-dimensional multipolygon.
```

6.12.2.21. ST_Point

This topic describes the Construct or function ST_Point and provides an example for reference.

Function declaration:

```
ST_Point(x, y)
ST_Point('point (x y)')
```

Description: This function constructs a two-dimensional point.

Example:

```
SELECT ST_Point(longitude, latitude) from src LIMIT 1;
SELECT ST_Point('point (0 0)') from src LIMIT 1;
-- Construct a two-dimensional point.
```

6.12.2.22. ST_PointFromWKB

This topic describes the Construct or function ST_PointFromWKB and provides an example for reference.

Function declaration:

```
ST_PointFromWKB(wkb)
```

Description: This function constructs a two-dimensional point based on the input well-known binary (WKB) expression that complies with the Open Geospatial Consortium (OGC) standards.

Example:

```
SELECT ST_PointFromWKB(ST_AsBinary(ST_GeomFromText('point (1 0)'))) FROM src LIMIT 1;
-- Construct a two-dimensional point.
```

6.12.2.23. ST_PointZ

This topic describes the Construct or function ST_PointZ and provides an example for reference.

Function declaration:

```
ST_PointZ(x, y, z)
```

Description: This function constructs a three-dimensional point.

Example:

```
SELECT ST_PointZ(longitude, latitude, elevation) from src LIMIT 1;
-- Construct a three-dimensional point.
```

6.12.2.24. ST_Polygon

This topic describes the Construct or function ST_Polygon and provides an example for reference.

Function declaration:

```
ST_Polygon(x, y, [x, y]*)
ST_Polygon('polygon( ... )')
```

Description: This function constructs a two-dimensional polygon.

Example:

```
SELECT ST_Polygon(1, 1, 1, 4, 4, 4, 4, 1) from src LIMIT 1;
-- Construct a square.
SELECT ST_Polygon('polygon ((1 1, 4 1, 1 4))') from src LIMIT 1;
-- Construct a triangle.
```

6.12.2.25. ST_PolyFromWKB

This topic describes the Construct or function ST_PolyFromWKB and provides an example for reference.

Function declaration:

```
ST_PolyFromWKB(wkb)
```

Description: This function constructs a two-dimensional polygon based on the input well-known binary (WKB) representation defined by the Open Geospatial Consortium (OGC).

Example:

```
SELECT ST_PolyFromWKB(ST_AsBinary(ST_GeomFromText('polygon ((0 0, 10 0, 0 10, 0 0))'))) FROM src LIMIT 1;
-- Construct a two-dimensional polygon.
```

6.12.2.26. ST_SetSRID

This topic describes the Constructor function ST_SetSRID and provides an example for reference.

Function declaration:

```
ST_SetSRID(<ST_Geometry>, SRID)
```

Description: This function sets the spatial reference system identifier (SRID) of the input geometry.

Example:

```
SELECT ST_SetSRID(ST_Point(1.5, 2.5), 4326) FROM src LIMIT 1;
-- Construct a point and set its SRID to 4326.
```

6.12.3. Accessors

6.12.3.1. ST_Area

This topic describes the Accessor function ST_Area and provides an example for reference.

Function declaration:

```
ST_Area(ST_Polygon)
```

Description: This function returns the areas of one or more polygons.

Example:

```
SELECT ST_Area(ST_Polygon(1,1, 1,4, 4,4, 4,1)) FROM src LIMIT 1;
```

Returned results:

```
9.0
```

6.12.3.2. ST_Centroid

This topic describes the Accessor function ST_Centroid and provides examples for reference.

Function declaration:

```
ST_Centroid(polygon)
```

Description: This function returns the centroid of the minimum bounding rectangle of the input polygon.

Examples:

```
SELECT ST_Centroid(ST_GeomFromText('polygon ((0 0, 3 6, 6 0, 0 0))')) FROM src LIMIT 1;
```

Returned results:

```
POINT(3 3)
```

```
SELECT ST_Centroid(ST_GeomFromText('polygon ((0 0, 0 8, 8 0, 0 0))')) FROM src LIMIT 1;
```

Returned results:

```
POINT(4 4)
```

6.12.3.3. ST_CoordDim

This topic describes the Accessor function `ST_CoordDim` and provides examples for reference.

Function declaration:

```
ST_CoordDim(geometry)
```

Description: This function returns the coordinate dimension of the input geometry.

Examples:

```
SELECT ST_CoordDim(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
2
```

```
SELECT ST_CoordDim(ST_PointZ(1.5,2.5, 3)) FROM src LIMIT 1;
```

Returned results:

```
3
```

```
SELECT ST_CoordDim(ST_Point(1.5, 2.5, 3., 4.)) FROM src LIMIT 1;
```

Returned results:

```
4
```

6.12.3.4. ST_Dimension

This topic describes the Accessor function `ST_Dimension` and provides examples for reference.

Function declaration:

```
ST_Dimension(geometry)
```

Description: This function returns the spatial dimension of the input geometry.

Examples:

```
SELECT ST_Dimension(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
0
```

```
SELECT ST_Dimension(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
1
```

```
SELECT ST_Dimension(ST_Polygon(2,0, 2,3, 3,0)) FROM src LIMIT 1;
```

Returned results:

```
2
```

6.12.3.5. ST_Distance

This topic describes the Accessor function `ST_Distance` and provides an example for reference.

Function declaration:

```
ST_Distance(ST_Geometry1, ST_Geometry2)
```

Description: This function returns the distance between two geometries.

Example:

```
SELECT ST_Distance(ST_Point(0.0,0.0), ST_Point(3.0,4.0)) FROM src LIMIT 1;
```

Returned results:

```
5.0
```

6.12.3.6. ST_GeodesicLengthWGS84

This topic describes the Accessor function `ST_GeodesicLengthWGS84` and provides examples for reference.

Function declaration:

```
ST_GeodesicLengthWGS84(line)
```

Description: This function returns the distance in meters on a spheroid based on World Geodetic System 1984 (WGS84). The geometry must be in WGS84. Otherwise, this function returns NULL.

Examples:

```
SELECT ST_GeodesicLengthWGS84(ST_SetSRID(ST_LineString(0.0,0.0, 0.3,0.4), 4326)) FROM src LIMIT 1;
```

Returned results:

```
55km
```

```
SELECT ST_GeodesicLengthWGS84(ST_GeomFromText('MultiLineString((0.0 80.0, 0.3 80.4))', 4326)) FROM src LIMIT 1;
```

Returned results:

```
45km
```

6.12.3.7. ST_GeometryN

This topic describes the Accessor function ST_GeometryN and provides examples for reference.

Function declaration:

```
ST_GeometryN(ST_GeometryCollection, n)
```

Description: This function returns the n^{th} geometry in the input geometry collection. n starts from 1.

Examples:

```
SELECT ST_GeometryN(ST_GeomFromText('multipoint ((10 40), (40 30), (20 20), (30 10))'), 3) FROM src LIMIT 1;
```

Returned results:

```
ST_Point(20 20)
```

```
SELECT ST_GeometryN(ST_GeomFromText('multilinestring ((2 4, 10 10), (20 20, 7 8))'), 2) FROM src LIMIT 1;
```

Returned results:

```
ST_Linestring(20 20, 7 8)
```

6.12.3.8. ST_Is3D

This topic describes the Accessor function ST_Is3D and provides an example for reference.

Function declaration:

```
ST_Is3D(geometry)
```

Description: If the input geometry has Z coordinates, this function returns True. Otherwise, this function returns False.

Example:

```
SELECT ST_Is3D(ST_Polygon(1,1, 1,4, 4,4, 4,1)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_Is3D(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_Is3D(ST_Point(3., 4.)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_Is3D(ST_PointZ(3., 4., 2)) FROM src LIMIT 1;
-- True is returned.
```

6.12.3.9. ST_IsClosed

This topic describes the Accessor function `ST_IsClosed` and provides an example for reference.

Function declaration:

```
ST_IsClosed(ST_[Multi]LineString)
```

Description: If the input linestring or linestrings are closed, this function returns True. Otherwise, it returns False.

Example:

```
SELECT ST_IsClosed(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- True is returned.
SELECT ST_IsClosed(ST_LineString(0.,0., 3.,4.)) FROM src LIMIT 1;
-- False is returned.
```

6.12.3.10. ST_IsEmpty

This topic describes the Accessor function `ST_IsEmpty` and provides an example for reference.

Function declaration:

```
ST_IsEmpty(geometry)
```

Description: If the input geometry is empty, this function returns True. Otherwise, it returns False.

Example:

```
SELECT ST_IsEmpty(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_IsEmpty(ST_GeomFromText('point empty')) FROM src LIMIT 1;
-- True is returned.
```

6.12.3.11. ST_IsMeasured

This topic describes the Accessor function `ST_IsMeasured` and provides an example for reference.

Function declaration:

```
ST_IsMeasured(geometry)
```

Description: If the input geometry has M coordinates (measures), this function returns True. Otherwise, it returns False.

Example:

```
SELECT ST_IsMeasured(ST_Polygon(1,1, 1,4, 4,4, 4,1)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_IsMeasured(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_IsMeasured(ST_Point(3., 4.)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_IsMeasured(ST_PointM(3., 4., 2)) FROM src LIMIT 1;
-- True is returned.
```

6.12.3.12. ST_IsSimple

This topic describes the Accessor function `ST_IsSimple` and provides an example for reference.

Function declaration:

```
ST_IsSimple(geometry)
```

Description: If the input geometry is a simple object, this function returns True. Otherwise, it returns False.

Example:

```
SELECT ST_IsSimple(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- True is returned.
SELECT ST_IsSimple(ST_LineString(0.,0., 1.,1., 0.,1., 1.,0.)) FROM src LIMIT 1;
-- False is returned.
```

6.12.3.13. ST_IsRing

This topic describes the Accessor function `ST_IsRing` and provides an example for reference.

Function declaration:

```
ST_IsRing(ST_LineString)
```

Description: This function returns True if the line string is a simple object or closed. Otherwise, it returns False.

Example:

```
SELECT ST_IsRing(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- True is returned.
SELECT ST_IsRing(ST_LineString(0.,0.,1.,1., 1.,2., 2.,1., 1.,1., 0.,0.)) FROM src LIMIT 1;
-- False is returned.
SELECT ST_IsRing(ST_LineString(0.,0., 3.,4.)) FROM src LIMIT 1;
-- False is returned.
```

6.12.3.14. ST_Length

This topic describes the Accessor function `ST_Length` and provides an example for reference.

Function declaration:

```
ST_Length(line)
```

Description: This function returns the length of the input line segment.

Example:

```
SELECT ST_Length(ST_Line(0.0,0.0, 3.0,4.0)) FROM src LIMIT 1;
```

Returned results:

```
5.0
```

6.12.3.15. ST_M

This topic describes the Accessor function ST_M and provides an example for reference.

Function declaration:

```
ST_M(geometry)
```

Description: This function returns the M coordinate of the input geometry.

Example:

```
SELECT ST_M(ST_PointM(3., 4., 2)) FROM src LIMIT 1;
```

Returned results:

```
2
```

6.12.3.16. ST_MaxM

This topic describes the Accessor function ST_MaxM and provides examples for reference.

Function declaration:

```
ST_MaxM(geometry)
```

Description: This function returns the maximum M coordinate of the input geometry.

Examples:

```
SELECT ST_MaxM(ST_PointM(1.5, 2.5, 2)) FROM src LIMIT 1;
```

Returned results:

```
2
```

```
SELECT ST_MaxM(ST_LineString('linestring m (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
```

Returned results:

```
1
```

6.12.3.17. ST_MinM

This topic describes the Accessor function ST_MinM and provides examples for reference.

Function declaration:

```
ST_MinM(geometry)
```

Description: This function returns the minimum M coordinate of the input geometry.

Examples:

```
SELECT ST_MinM(ST_PointM(1.5, 2.5, 2)) FROM src LIMIT 1;
```

Returned results:

```
2
```

```
SELECT ST_MinM(ST_LineString('linestring m (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
```

Returned results:

```
1
```

6.12.3.18. ST_X

This topic describes the Accessor function ST_X and provides an example for reference.

Function declaration:

```
ST_X(point)
```

Description: This function returns the X coordinate of the input point.

Example:

```
SELECT ST_X(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
1.5
```

6.12.3.19. ST_Y

This topic describes the Accessor function ST_Y and provides an example for reference.

Function declaration:

```
ST_Y(point)
```

Description: This function returns the Y coordinate of the input point.

Example:

```
SELECT ST_Y(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
2.5
```

6.12.3.20. ST_Z

This topic describes the Accessor function ST_Z and provides an example for reference.

Function declaration:

```
ST_Z(point)
```

Description: This function returns the Z coordinate of the input point.

Example:

```
SELECT ST_Z(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
1.5
```

6.12.3.21. ST_MaxX

This topic describes the Accessor function ST_MaxX and provides examples for reference.

Function declaration:

```
ST_MaxX(geometry)
```

Description: This function returns the maximum X coordinate of the input geometry.

Examples:

```
SELECT ST_MaxX(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
1.5
```

```
SELECT ST_MaxX(ST_LineString(1.5,2.5,3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
3.0
```

6.12.3.22. ST_MaxY

This topic describes the Accessor function ST_MaxX and provides examples for reference.

Function declaration:

```
ST_MaxY(geometry)
```

Description: This function returns the maximum Y coordinate of the input geometry.

Examples:

```
SELECT ST_MaxY(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
2.5
```

```
SELECT ST_MaxY(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
2.5
```

6.12.3.23. ST_MaxZ

This topic describes the Accessor function ST_MaxZ and provides examples for reference.

Function declaration:

```
ST_MaxZ(geometry)
```

Description: This function returns the maximum Z coordinate of the input geometry.

Examples:

```
SELECT ST_MaxZ(ST_PointZ(1.5, 2.5, 2)) FROM src LIMIT 1;
```

Returned results:

```
2
```

```
SELECT ST_MaxZ(ST_LineString('linestring z (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
```

Returned results:

```
1
```

6.12.3.24. ST_MinX

This topic describes the Accessor function ST_MinX and provides examples for reference.

Function declaration:

```
ST_MinX(geometry)
```

Description: This function returns the minimum X coordinate of the input geometry.

Examples:

```
SELECT ST_MinX(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
1.5
```

```
SELECT ST_MinX(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
3.0
```

6.12.3.25. ST_MinY

This topic describes the Accessor function ST_MinY and provides examples for reference.

Function declaration:

```
ST_MinY(geometry)
```

Description: This function returns the minimum Y coordinate of the input geometry.

Examples:

```
SELECT ST_MinY(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
2.5
```

```
SELECT ST_MinY(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
2.2
```

6.12.3.26. ST_MinZ

This topic describes the Accessor function ST_MinZ and provides examples for reference.

Function declaration:

```
ST_MinZ(geometry)
```

Description: This function returns the minimum Z coordinate of the input geometry.

Examples:

```
SELECT ST_MinZ(ST_PointZ(1.5, 2.5, 2)) FROM src LIMIT 1;
```

Returned results:

```
2
```

```
SELECT ST_MinZ(ST_LineString('linestring z (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
```

Returned results:

```
1
```

6.12.3.27. ST_NumGeometries

This topic describes the Accessor function ST_NumGeometries and provides examples for reference.

Function declaration:

```
ST_NumGeometries(ST_GeometryCollection)
```

Description: This function returns the number of geometries in the input geometry collection.

Examples:

```
SELECT ST_NumGeometries(ST_GeomFromText('multipoint ((10 40), (40 30), (20 20), (30 10))')) FROM src  
LIMIT 1;
```

Returned results:

```
4
```

```
SELECT ST_NumGeometries(ST_GeomFromText('multilinestring ((2 4, 10 10), (20 20, 7 8))')) FROM src LI  
MIT 1;
```

Returned results:

```
2
```

6.12.3.28. ST_NumInteriorRing

This topic describes the Accessor function ST_NumInteriorRing and provides examples for reference.

Function declaration:

```
ST_NumInteriorRing(ST_Polygon)
```

Description: This function returns the number of interior rings of the input polygon.

Examples:

```
SELECT ST_NumInteriorRing(ST_Polygon(1,1, 1,4, 4,1)) FROM src LIMIT 1;
```

Returned results:

```
0
```

```
SELECT ST_NumInteriorRing(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1 1))')) FROM s  
rc LIMIT 1;
```

Returned results:

```
1
```

6.12.3.29. ST_NumPoints

This topic describes the Accessor function ST_NumPoints and provides examples for reference.

Function declaration:

```
ST_NumPoints(geometry)
```

Description: This function returns the number of points in the input geometry.

Examples:

```
SELECT ST_NumPoints(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
1
```

```
SELECT ST_NumPoints(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
2
```

```
SELECT ST_NumPoints(ST_GeomFromText('polygon ((0 0, 10 0, 0 10, 0 0))')) FROM src LIMIT 1;
```

Returned results:

```
4
```

6.12.3.30. ST_PointN

This topic describes the Accessor function ST_PointN and provides an example for reference.

Function declaration:

```
ST_PointN(ST_Geometry, n)
```

Description: This function returns the n^{th} point of one or more linestrings.

Example:

```
SELECT ST_PointN(ST_LineString(1.5,2.5, 3.0,2.2), 2) FROM src LIMIT 1;
```

Returned results:

```
POINT(3.0 2.2)
```

6.12.3.31. ST_StartPoint

This topic describes the Accessor function `ST_StartPoint` and provides an example for reference.

Function declaration:

```
ST_StartPoint(geometry)
```

Description: This function returns the first point of the input linestring.

Example:

```
SELECT ST_StartPoint(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
POINT(1.5 2.5)
```

6.12.3.32. ST_EndPoint

This topic describes the Accessor function `ST_EndPoint` and provides an example for reference.

Function declaration:

```
ST_EndPoint(geometry)
```

Description: This function returns the last point of the input linestring.

Example:

```
SELECT ST_EndPoint(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
```

Returned results:

```
POINT(3.0 2.0)
```

6.12.3.33. ST_SRID

This topic describes the Accessor function `ST_SRID` and provides an example for reference.

Function declaration:

```
ST_SRID(ST_Geometry)
```

Description: This function returns the spatial reference system identifier (SRID) of the input geometry.

Example:

```
SELECT ST_SRID(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
```

Returned results:

```
returns SRID 0
```

6.12.4. Operations

6.12.4.1. ST_Aggr_ConvexHull

This topic describes the Operation function `ST_Aggr_ConvexHull` and provides an example for reference.

Function declaration:

```
ST_Aggr_ConvexHull(ST_Geometry)
```

Description: This function returns a convex hull for input geometries by using aggregate transformation.

Example:

```
SELECT ST_Aggr_ConvexHull(geometry) FROM source;
-- Return the convex hull of input geometries from the data source by using aggregate transformation.
```

6.12.4.2. ST_Aggr_Intersection

This topic describes the Operation function `ST_Aggr_Intersection` and provides an example for reference.

Function declaration:

```
ST_Aggr_Intersection(ST_Geometry)
```

Description: This function returns the intersection of input geometries by using aggregate transformation.

Example:

```
SELECT ST_Aggr_Intersection(geometry) FROM source;
-- Return the intersection of input geometries from the data source by using aggregate transformation.
```

6.12.4.3. ST_Aggr_Union

This topic describes the Operation function `ST_Aggr_Union` and provides an example for reference.

Function declaration:

```
ST_Aggr_Union(ST_Geometry)
```

Description: This function returns a union of input geometries by using aggregate transformation.

Example:

```
SELECT ST_Aggr_Union(geometry) FROM source;
-- Return the union of input geometries from the data source by using aggregate transformation.
```

6.12.4.4. ST_Bin

This topic describes the Operation function `ST_Bin`.

Function declaration:

```
ST_Bin(placeholder)
```

Description: This function returns the bin ID of the input point.

6.12.4.5. ST_BinEnvelope

This topic describes the Operation function ST_BinEnvelope.

Function declaration:

```
ST_BinEnvelope(binsize, point)
```

Description: This function returns the binary envelope for the input point.

Function declaration:

```
ST_BinEnvelope(binsize, binid)
```

Description: This function returns the binary envelope for the input bin ID.

6.12.4.6. ST_Boundary

This topic describes the Operation function ST_Boundary and provides examples for reference.

Function declaration:

```
ST_Boundary(ST_Geometry)
```

Description: This function returns the boundary of the input geometry ST_Geometry.

Examples:

```
SELECT ST_Boundary(ST_LineString(0,1, 1,0)) FROM src LIMIT 1;
```

Returned results:

```
MULTIPOINT((1 0), (0 1))
```

```
SELECT ST_Boundary(ST_Polygon(1,1, 4,1, 1,4)) FROM src LIMIT 1;
```

Returned results:

```
LINESTRING(1 1, 4 1, 1 4, 1 1)
```

6.12.4.7. ST_Buffer

This topic describes the Operation function ST_Buffer.

Function declaration:

```
ST_Buffer(geometry, distance)
```

Description: This function returns a geometry that indicates all points whose distance from this geometry to the input geometry is less than or equal to the value of the distance parameter.

6.12.4.8. ST_ConvexHull

This topic describes the Operation function ST_ConvexHull and provides an example for reference.

Function declaration:

```
ST_ConvexHull(ST_Geometry, ST_Geometry, ...)
```

Description: This function returns the geometric object ST_Geometry as the convex hull of the specified geometry.

Example:

```
SELECT ST_AsText(ST_ConvexHull(ST_Point(0, 0), ST_Point(0, 1), ST_Point(1, 1))) FROM onerow;
```

Returned results:

```
MULTIPOLYGON (((0 0, 1 1, 0 1, 0 0)))
```

6.12.4.9. ST_Difference

This topic describes the Operation function ST_Difference and provides examples for reference.

Function declaration:

```
ST_Difference(ST_Geometry1, ST_Geometry2)
```

Description: This function returns a geometry that indicates the difference between ST_Geometry1 and ST_Geometry2.

Examples:

```
SELECT ST_AsText(ST_Difference(ST_MultiPoint(1, 1, 1.5, 1.5, 2, 2), ST_Point(1.5, 1.5))) FROM onerow ;
```

Returned results:

```
MULTIPOINT (1 1, 2 2)
```

```
SELECT ST_AsText(ST_Difference(ST_Polygon(0, 0, 0, 10, 10, 10, 10, 0), ST_Polygon(0, 0, 0, 5, 5, 5, 5, 0))) from onerow;
```

Returned results:

```
MULTIPOLYGON (((10 0, 10 10, 0 10, 0 5, 5 5, 5 0, 10 0)))
```

6.12.4.10. ST_Envelope

This topic describes the Operation function ST_Envelope and provides examples for reference.

Function declaration:

```
ST_Envelope(ST_Geometry)
```

Description: This function returns the envelope of the input geometry. If the specified geometry is a point, a horizontal line, or a vertical line, this function returns the common difference or an empty envelope.

Examples:

```
SELECT ST_Envelope(ST_LineString(0,0, 2,2)) from src LIMIT 1;
```

Returned results:

```
POLYGON ((0 0, 2 0, 2 2, 0 2, 0 0))
```

```
SELECT ST_Envelope(ST_Polygon(2,0, 2,3, 3,0)) from src LIMIT 1;
```

Returned results:

```
POLYGON ((2 0, 3 0, 3 3, 2 3, 2 0))
```

6.12.4.11. ST_ExteriorRing

This topic describes the Operation function ST_ExteriorRing and provides examples for reference.

Function declaration:

```
ST_ExteriorRing(polygon)
```

Description: This function returns the exterior ring of a polygon as a linestring.

Examples:

```
SELECT ST_ExteriorRing(ST_Polygon(1,1, 1,4, 4,1)) FROM src LIMIT 1;
```

Returned results:

```
LINestring(1 1, 4 1, 1 4, 1 1)
```

```
SELECT ST_ExteriorRing(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1 1))')) FROM src  
LIMIT 1;
```

Returned results:

```
LINestring (8 0, 0 8, 0 0, 8 0)
```

6.12.4.12. ST_InteriorRingN

This topic describes the Operation function ST_InteriorRingN and provides an example for reference.

Function declaration:

```
ST_InteriorRingN(ST_Polygon, n)
```

Description: This function returns the n^{th} interior ring of a polygon as a linestring.

Example:

```
SELECT ST_InteriorRingN(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1 1))'), 1) FROM
src LIMIT 1;
```

Returned results:

```
LINESTRING (1 1, 5 1, 1 5, 1 1)
```

6.12.4.13. ST_Intersection

This topic describes the Operation function `ST_Intersection` and provides examples for reference.

Function declaration:

```
ST_Intersection(ST_Geometry1, ST_Geometry2)
```

Description: This function returns a geometry that indicates the intersection of the input geometries. If the input geometries intersect in a lower dimension, `ST_Intersection` may drop lower-dimension intersections or return a closed linestring.

Examples:

```
SELECT ST_AsText(ST_Intersection(ST_Point(1,1), ST_Point(1,1))) FROM onerow;
```

Returned results:

```
POINT (1 1)
```

```
SELECT ST_AsText(ST_Intersection(ST_GeomFromText('linestring(0 2, 0 0, 2 0)'), ST_GeomFromText('line
string(0 3, 0 1, 1 0, 3 0)'))) FROM onerow;
```

Returned results:

```
MULTILINESTRING ((1 0, 2 0), (0 2, 0 1))
```

```
SELECT ST_AsText(ST_Intersection(ST_LineString(0,2, 2,3), ST_Polygon(1,1, 4,1, 4,4, 1,4))) FROM oner
ow;
```

Returned results:

```
MULTILINESTRING ((1 2.5, 2 3))
```

```
SELECT ST_AsText(ST_Intersection(ST_Polygon(2,0, 2,3, 3,0), ST_Polygon(1,1, 4,1, 4,4, 1,4))) FROM on
erow;
```

Returned results:

```
MULTIPOLYGON (((2.67 1, 2 3, 2 1, 2.67 1)))
```

```
SELECT ST_AsText(ST_Intersection(ST_Polygon(2,0, 3,1, 2,1), ST_Polygon(1,1, 4,1, 4,4, 1,4))) FROM on
erow;
```

Returned results:

```
MULTIPOLYGON EMPTY or LINESTRING (2 1, 3 1, 2 1)
```

6.12.4.14. ST_SymmetricDiff

This topic describes the ST_SymmetricDiff function and provides examples of using this function.

Function declaration:

```
ST_SymmetricDiff(ST_Geometry1, ST_Geometry2)
```

Description: This function returns a geometry that consists of the symmetric differences of the input geometries.

Examples:

```
SELECT ST_AsText(ST_SymmetricDiff(ST_LineString('linestring(0 2, 2 2)'), ST_LineString('linestring(1 2, 3 2)'))) FROM onerow;
```

Returned result:

```
MULTILINESTRING((0 2, 1 2), (2 2, 3 2))
```

```
SELECT ST_AsText(ST_SymmetricDiff(ST_SymmetricDiff(ST_Polygon('polygon((0 0, 2 0, 2 2, 0 2, 0 0)'), ST_Polygon('polygon((1 1, 3 1, 3 3, 1 3, 1 1)')))) FROM onerow;
```

Returned result:

```
MULTIPOLYGON (((0 0, 2 0, 2 1, 1 1, 1 2, 0 2, 0 0)), ((3 1, 3 3, 1 3, 1 2, 2 2, 2 1, 3 1)))
```

6.12.4.15. ST_Union

This topic describes the ST_Union function and provides an example of using this function.

Function declaration:

```
ST_Union(ST_Geometry, ST_Geometry, ...)
```

Description: This function returns a geometry that is the union of the input geometries.

Example:

```
SELECT ST_AsText(ST_Union(ST_Polygon(1, 1, 1, 4, 4, 4, 4, 1), ST_Polygon(4, 1, 4, 4, 4, 8, 8, 1))) FROM onerow;
```

Returned result:

```
MULTIPOLYGON (((4 1, 8 1, 4 8, 4 4, 1 4, 1 1, 4 1)))
```

6.12.5. Relationship tests

6.12.5.1. ST_Contains

This topic describes the ST_Contains function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Contains(geometry1, geometry2)
```

Description: If geometry1 contains geometry2, this function returns true. Otherwise, this function returns false.

Example:

```
SELECT ST_Contains(st_polygon(1,1, 1,4, 4,4, 4,1), st_point(2, 3) from src LIMIT 1;
-- true is returned.
SELECT ST_Contains(st_polygon(1,1, 1,4, 4,4, 4,1), st_point(8, 8) from src LIMIT 1;
-- false is returned.
```

6.12.5.2. ST_Crosses

This topic describes the ST_Crosses function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Crosses(geometry1, geometry2)
```

Description: If geometry1 crosses geometry2, this function returns true. Otherwise, this function returns false.

 **Note** Crossing indicates that some points in the two geometries are the same.

Example:

```
SELECT ST_Crosses(st_linestring(0,0, 1,1), st_linestring(1,0, 0,1)) from src LIMIT 1;
-- true is returned.
SELECT ST_Crosses(st_linestring(2,0, 2,3), st_polygon(1,1, 1,4, 4,4, 4,1)) from src LIMIT 1;
-- true is returned.
SELECT ST_Crosses(st_linestring(0,2, 0,1), ST_linestring(2,0, 1,0)) from src LIMIT 1;
-- false is returned.
```

6.12.5.3. ST_Disjoint

This topic describes the ST_Disjoint function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Disjoint(geometry1, geometry2)
```

Description: If geometry1 and geometry2 do not intersect, this function returns true. Otherwise, this function returns false.

Example:

```
SELECT ST_Disjoint(ST_LineString(0,0, 0,1), ST_LineString(1,1, 1,0)) from src LIMIT 1;
-- true is returned.
SELECT ST_Disjoint(ST_LineString(0,0, 1,1), ST_LineString(1,0, 0,1)) from src LIMIT 1;
-- false is returned.
```

6.12.5.4. ST_EnvIntersects

This topic describes the ST_EnvIntersects function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_EnvIntersects(ST_Geometry1, ST_Geometry2)
```

Description: If the envelopes of ST_Geometry1 and ST_Geometry2 intersect, this function returns true. Otherwise, this function returns false.

Sample statements:

```
SELECT ST_EnvIntersects(ST_LineString(0,0, 1,1), ST_LineString(1,3, 2,2)) from src LIMIT 1;
-- false is returned.
SELECT ST_EnvIntersects(ST_LineString(0,0, 2,2), ST_LineString(1,0, 3,2)) from src LIMIT 1;
-- true is returned.
```

6.12.5.5. ST_Equals

This topic describes the ST_Equals function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Equals(geometry1, geometry2)
```

Description: If geometry1 equals geometry2, this function returns true. Otherwise, this function returns false.

Example:

```
SELECT ST_Equals(st_linestring(0,0, 1,1), st_linestring(1,1, 0,0)) from src LIMIT 1;
-- true is returned.
SELECT ST_Equals(st_linestring(0,0, 1,1), st_linestring(1,0, 0,1)) from src LIMIT 1;
-- false is returned.
```

6.12.5.6. ST_Intersects

This topic describes the ST_Intersects function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Intersects(geometry1, geometry2)
```

Description: If geometry1 and geometry2 intersect, this function returns true. Otherwise, this function returns false.

Example:

```
SELECT ST_Intersects(st_linestring(0,0, 1,1), st_linestring(1,1, 0,0)) from src LIMIT 1;
-- true is returned.
SELECT ST_Intersects(st_linestring(0,0, 1,1), st_linestring(1,0, 0,1)) from src LIMIT 1;
-- true is returned.
SELECT ST_Intersects(ST_LineString(2,0, 2,3), ST_Polygon(1,1, 4,1, 4,4, 1,4)) from src LIMIT 1;
-- true is returned.
SELECT ST_Intersects(ST_LineString(8,7, 7,8), ST_Polygon(1,1, 4,1, 4,4, 1,4)) from src LIMIT 1;
-- false is returned.
```

6.12.5.7. ST_Overlaps

This topic describes the ST_Overlaps function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Overlaps(geometry1, geometry2)
```

Description: If geometry1 and geometry2 overlap, this function returns true. Otherwise, this function returns false. Overlapping excludes the tangency of the geometries.

Example:

```
SELECT ST_Overlaps(st_polygon(2,0, 2,3, 3,0), st_polygon(1,1, 1,4, 4,4, 4,1)) from src LIMIT 1;
-- true is returned.
SELECT ST_Overlaps(st_polygon(2,0, 2,1, 3,1), ST_Polygon(1,1, 1,4, 4,4, 4,1)) from src LIMIT 1;
-- false is returned.
```

6.12.5.8. ST_Relate

This topic describes the ST_Relate function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Relate(geometry1, geometry2)
```

Description: If geometry1 has the specified Dimensionally Extended nine-Intersection Model (DE-9IM) relationship with geometry2, this function returns true. Otherwise, this function returns false.

Example:

```
SELECT ST_Relate(st_polygon(2,0, 2,1, 3,1), ST_Polygon(1,1, 1,4, 4,4, 4,1), '****T****') from src LI
MIT 1;
-- true is returned.
SELECT ST_Relate(st_polygon(2,0, 2,1, 3,1), ST_Polygon(1,1, 1,4, 4,4, 4,1), 'T*****') from src LI
MIT 1;
-- false is returned.
SELECT ST_Relate(st_linestring(0,0, 3,3), ST_linestring(1,1, 4,4), '*****') from src LIMIT 1;
-- true is returned.
SELECT ST_Relate(st_linestring(0,0, 3,3), ST_linestring(1,1, 4,4), '****T****') from src LIMIT 1;
-- false is returned.
```

6.12.5.9. ST_Touches

This topic describes the ST_Touches function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Touches(geometry1, geometry2)
```

Description: If geometry1 and geometry2 spatially touch and have no similar interior points, this function returns true. Otherwise, this function returns false.

Example:

```
SELECT ST_Touches(st_point(1, 2), st_polygon(1, 1, 1, 4, 4, 4, 4, 1)) from src LIMIT 1;
-- true is returned.
SELECT ST_Touches(st_point(8, 8), st_polygon(1, 1, 1, 4, 4, 4, 4, 1)) from src LIMIT 1;
-- false is returned.
```

6.12.5.10. ST_Within

This topic describes the ST_Within function and provides an example of using this function.

Function declaration:

```
BOOLEAN ST_Within(geometry1, geometry2)
```

Description: If geometry1 is within geometry2, this function returns true. Otherwise, this function returns false.

Example:

```
SELECT ST_Within(st_point(2, 3), st_polygon(1,1, 1,4, 4,4, 4,1)) from src LIMIT 1;
-- true is returned.
SELECT ST_Within(st_point(8, 8), st_polygon(1,1, 1,4, 4,4, 4,1)) from src LIMIT 1;
-- false is returned.
```

6.12.6. Geohash index functions

6.12.6.1. ST_GeoHash

This topic describes the ST_GeoHash function and provides an example of using this function.

Function declaration:

```
string ST_GeoHash(st_geometry geometry, integer precision=full_precision)
string ST_GeoHash(double longitude, double latitude, integer precision=full_precision)
```

Description: This function returns the unique Geohash string of the specified point. This function uses a function with the ST_ prefix or the specified longitudes and latitudes as input parameters. If the precision parameter is not specified, the maximum precision is used.

Example:

```
SELECT ST_GeoHash(ST_Point(-102.849854, 36.451113), 8);
SELECT ST_GeoHash(ST_GeomFromText('POINT(-102.849854 36.451113)'));
SELECT ST_GeoHash(-102.849854, 36.451113, 10);
```

6.12.6.2. ST_PointFromGeoHash

This topic describes the ST_PointFromGeoHash function and provides an example of using this function.

Function declaration:

```
st_geometry ST_PointFromGeoHash(string geohash, integer precision=full_precision)
```

Description: This function returns a point based on the input Geohash value. If the precision parameter is not specified, the maximum precision is used.

Example:

```
SELECT ST_AsText(ST_PointFromGeoHash('9wqz7eep0eyq'));
SELECT ST_AsText(ST_PointFromGeoHash('9wqz7eep0eyq', 4));
```

6.12.6.3. ST_EnvelopeFromGeoHash

This topic describes the ST_EnvelopeFromGeoHash function and provides an example of using this function.

Function declaration:

```
st_geometry ST_EnvelopeFromGeoHash(string geohash, integer precision=full_precision)
```

Description: This function returns the envelope of the specified precision based on the input Geohash value. If the precision parameter is not specified, the maximum precision is used.

Example:

```
SELECT ST_AsText(ST_EnvelopeFromGeoHash('9wqz7eep0eyq', 8));
SELECT ST_AsText(ST_EnvelopeFromGeoHash('9wqz7eep0eyq'));
```

6.12.6.4. ST_GeoHashNeighbours

This topic describes the ST_GeoHashNeighbours function and provides an example of using this function.

Function declaration:

```
list_of_string ST_GeoHashNeighbours(double longitude, double latitude, integer precision)
```

Description: This function is a user-defined table-valued function (UDTF) that generates nine data records. This function returns nine Geohash strings of the current point and its eight neighboring points based on the input longitude, latitude, and precision. These parameters must be specified.

Example:

```
SELECT ST_GeoHashNeighbours(-102.849854, 36.451113, 10);
```

6.12.7. S2 mesh functions

6.12.7.1. ST_S2CellIdsFromGeom

This topic describes the ST_S2CellIdsFromGeom function and provides an example of using this function.

Function declaration:

```
list_of_string ST_S2CellIdsFromGeom(st_geometry geometry, integer level)
```

Description: This function overwrites the input geometry by using S2 cells at the specified level. Then, it returns the IDs of all S2 cells.

Example:

```
SELECT ST_S2CellIdsFromGeom(ST_Point(-102.849854, 36.451113), 4);
SELECT ST_S2CellIdsFromGeom(ST_LineString('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)'), 17) as cellid;
```

6.12.7.2. ST_S2CellIdsFromText

This topic describes the ST_S2CellIdsFromText function and provides an example of using this function.

Function declaration:

```
list_of_string ST_S2CellIdsFromText(string wkt, integer level)
```

Description: This function overwrites the well-known text (WKT) representation of the input geometry by using S2 cells at the specified level. Then, it returns the IDs of all S2 cells.

Example:

```
SELECT ST_S2CellIdsFromText(ST_GeomFromText('POINT(-102.849854 36.451113)'), 4);
SELECT ST_S2CellIdsFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)', 17) as cellid;
```

6.12.7.3. ST_S2CellCenterPoint

This topic describes the ST_S2CellCenterPoint function and provides an example of using this function.

Function declaration:

```
st_point ST_S2CellCenterPoint(string cellId)
```

Description: This function calculates the center point of the cell specified by the cellId parameter in the input S2 cell.

Example:

```
SELECT ST_S2CellCenterPoint('549015');
SELECT ST_AsText(ST_S2CellCenterPoint('89e37f091'));
```

6.12.7.4. ST_S2CellNeighbours

This topic describes the ST_S2CellNeighbours function and provides an example of using this function.

Function declaration:

```
list_of_string ST_S2CellNeighbours(string cellId, integer level)
```

Description: This function calculates the neighboring S2 cells of the cell specified by the cellId parameter at the specified level. Then, it returns the IDs of all neighboring S2 cells.

Example:

```
SELECT ST_S2CellNeighbours('549015', 10);
SELECT ST_S2CellNeighbours('89e37f091', 16) as neighbour;
```

6.12.8. Geodesic functions

6.12.8.1. ST_AreaWGS84

This topic describes the ST_AreaWGS84 function and provides an example of using this function.

Function declaration:

```
double ST_AreaWGS84(st_geometry geometry)
```

Description: This function returns the approximate geodesic area of the input geometry based on World Geodetic System 1984 (WGS84). This function converts the coordinates of the input geometry from EPSG:4326 to EPSG:3857. Then, it calculates the plane area in square meters.

Example:

```
SELECT ST_AreaWGS84(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450, 743265 2967450,743265.625 2967416,743238 2967416))'));
```

6.12.8.2. ST_DistanceWGS84

This topic describes the ST_DistanceWGS84 function and provides an example of using this function.

Function declaration:

```
double ST_DistanceWGS84(st_geometry geometry1, st_geometry geometry2)
```

Description: This function returns the approximate geodesic distance of the input geometry based on World Geodetic System 1984 (WGS84). This function converts the coordinates of the input geometry from EPSG:4326 to EPSG:3857. Then, it calculates the plane distance in meters.

Example:

```
SELECT ST_DistanceWGS84(ST_GeomFromText('POINT(-72.1235 42.3521)'), ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)'));
```

6.12.8.3. ST_BufferWGS84

This topic describes the ST_BufferWGS84 function and provides an example of using this function.

Function declaration:

```
st_geometry ST_BufferWGS84(st_geometry geometry, double radius)
```

Description: This function returns the approximate geodesic buffer of the input geometry based on World Geodetic System 1984 (WGS84). This function converts the coordinates of the input geometry from EPSG:4326 to EPSG:3857. Then, it calculates the plane buffer and converts the coordinates back to EPSG:4326.

Example:

```
SELECT ST_AsText(ST_BufferWGS84(ST_GeomFromText('POINT(-72.1235 42.3521)'), 10));
```

6.12.8.4. ST_GeodesicDistance

This topic describes the ST_GeodesicDistance function and provides an example of using this function.

Function declaration:

```
double ST_GeodesicDistance(double lon1, double lat1, double lon2, double lat2, string method = VINCENTY)
double ST_GeodesicDistance(st_geometry geo1, st_geometry geo2, string method = VINCENTY)
```

Description: This function calculates the geodesic distance between two points by using the specified method. The supported methods are Vincenty, LawOfCosines, and Haversine. The default value of the method parameter is VINCENTY. The return value is in radians.

Example:

```
SELECT ST_GeodesicDistance(ST_GeomFromText('POINT(152.352298 -24.875975)'), ST_GeomFromText('POINT(151.960336 -24.993289)'), 'LawOfCosines');
```

6.12.8.5. ST_Distance_Sphere

This topic describes the ST_Distance_Sphere function and provides an example of using this function.

Function declaration:

```
double ST_Distance_Sphere(st_point geo1, st_point2 geo2)
double ST_Distance_Sphere(double lng1, double lat1, double lng2, double lat2)
```

Description: This function uses the algorithm provided by AMAP to calculate the approximate geodesic distance between the two input points. This function uses ST_Point or the specified longitudes and latitudes as input parameters.

Example:

```
SELECT ST_Distance_Sphere(
    ST_GeomFromText('POINT(116.292078 39.919622)'),
    ST_GeomFromText('POINT(116.286676 39.919593)');
```

Returned result:

```
+-----+
| _c0   |
+-----+
| 460.6965312526471 |
+-----+
```

6.12.8.6. ST_Area_Sphere

This topic describes the ST_Area_Sphere function and provides an example of using this function.

Function declaration:

```
double ST_Area_Sphere(st_geometry geo)
```

Description: This function uses the algorithm provided by AMAP to calculate the geodesic area of the input geometry. This function uses only ST_Polygon and ST_MultiPolygon as input parameters.

Example:

```
SELECT geospatial.ST_Area_Sphere(geospatial.ST_GeomFromText('POLYGON((116.259097 40.202114,116.25902
4 40.20199,116.258768 40.201662,116.258376 40.201341,116.258031 40.201036,116.257675 40.200734,116.2
57429 40.200656,116.257357 40.200562,116.257392 40.200051,116.257506 40.199433,116.257569 40.198586,
116.257564 40.19756,116.257561 40.197372,116.257552 40.197036,116.257554 40.19675,116.257539 40.1966
47,116.257502 40.19653,116.257343 40.196389,116.257153 40.196276,116.256733 40.196071,116.25582 40.1
95646,116.255628 40.195611,116.255468 40.195653,116.255385 40.195742,116.255347 40.195849,116.255258
40.197143,116.255103 40.199576,116.255078 40.200585,116.251227 40.20059,116.251098 40.203978,116.259
433 40.204111,116.259247 40.203079,116.259097 40.202114)))');
```

Returned result:

```
+-----+
| _c0    |
+-----+
| 353493.765625 |
+-----+
```

6.12.9. R-tree index functions

6.12.9.1. ST_BuildRtreeIndex

This topic describes the ST_BuildRtreeIndex function and provides an example of using this function.

Function declaration:

```
RTree ST_BuildRtreeIndex(string uniqueId, string geometryWkt)
```

Description: This function is a user-defined aggregate function (UDAF). It uses the unique ID and well-known text (WKT) string of each geometry as input parameters to create the R-tree index. This function must be used with other R-tree functions.

Example:

```
SELECT geospatial.ST_BuildRtreeIndex(id, shape) AS index FROM poi_sample;
```

6.12.9.2. ST_ContainsFromRtree

This topic describes the ST_ContainsFromRtree function.

Function declaration:

```
ST_ContainsFromRtree(string uniqueId, string geometryWkt, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling ST_BuildRtreeIndex as input parameters. This function returns the IDs of objects that are contained by the geometry from the R-tree index. This function is used to accelerate the ST_Contains query.

6.12.9.3. ST_CrossesFromRtree

This topic describes the ST_CrossesFromRtree function.

Function declaration:

```
ST_CrossesFromRtree(string uniqueId, string geometryWkt, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling `ST_BuildRTreeIndex` as input parameters. This function returns the IDs of objects that cross the geometry from the R-tree index. This function is used to accelerate the `ST_Crosses` query.

6.12.9.4. ST_EqualsFromRTree

This topic describes the `ST_EqualsFromRTree` function.

Function declaration:

```
ST_EqualsFromRTree(string uniqueId, string geometryWkt, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling `ST_BuildRTreeIndex` as input parameters. This function returns the IDs of objects that equal the geometry from the R-tree index. This function is used to accelerate the `ST_Equals` query.

6.12.9.5. ST_IntersectsFromRTree

This topic describes the `ST_IntersectsFromRTree` function.

Function declaration:

```
ST_IntersectsFromRTree(string uniqueId, string geometryWkt, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling `ST_BuildRTreeIndex` as input parameters. This function returns the IDs of objects that intersect with the geometry from the R-tree index. This function is used to accelerate the `ST_Intersects` query.

6.12.9.6. ST_OverlapsFromRTree

This topic describes the `ST_OverlapsFromRTree` function.

Function declaration:

```
ST_OverlapsFromRTree(string uniqueId, string geometryWkt, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling `ST_BuildRTreeIndex` as input parameters. This function returns the IDs of objects that overlap with the geometry from the R-tree index. This function is used to accelerate the `ST_Overlaps` query.

6.12.9.7. ST_TouchesFromRTree

This topic describes the `ST_TouchesFromRTree` function.

Function declaration:

```
ST_TouchesFromRTree(string uniqueId, string geometryWkt, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling `ST_BuildRTreeIndex` as input parameters. This function returns the IDs of objects that spatially touch the geometry from the R-tree index. This function is used to accelerate the `ST_Touches` query.

6.12.9.8. ST_WithinFromRTree

This topic describes the ST_WithinFromRTree function and provides an example of using this function.

Function declaration:

```
ST_WithinFromRTree(string uniqueId, string geometryWkt, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling ST_BuildRTreeIndex as input parameters. This function returns the IDs of objects that include the geometry from the R-tree index. This function is used to accelerate the ST_Within query.

Example:

Query the intersections of line segments in the A table and polygons in the B table.

```
set odps.sql.allow.cartesian=true;
SELECT a.id as link_id, b.id as shape_id
FROM link_sample_wkt a, poi_sample_wkt b
WHERE geospatial.ST_IsValid(b.shape)
AND geospatial.ST_Intersects(
    geospatial.ST_LineString(a.line),
    geospatial.ST_Multipolygon(b.shape));
```

Returned result:

```
Summary:
resource cost: cpu 3.28 Core * Min, memory 5.76 GB * Min
inputs:
  meta_dev.poi_sample_wkt: 1000 (237592 bytes)
  meta_dev.link_sample_wkt: 1000 (105940 bytes)
outputs:
Job run time: 111.000
+-----+-----+
| link_id | shape_id |
+-----+-----+
| 5121371185457659960 | B000A844XK |
| 5121377123249946651 | B000A85TV4 |
| 5121377166199619654 | B000A844KT |
+-----+-----+
```

After optimization by using the new function:

```
SELECT /*+mapjoin(i)*/
    geospatial.ST_IntersectsFromRTree(id, line, i.index)
    AS (link_id, shape_id)
FROM link_sample_wkt
JOIN
(
    SELECT geospatial.ST_BuildRTreeIndex(id, shape) AS index
    FROM poi_sample_wkt
    WHERE geospatial.ST_IsValid(shape)
) i;
```

Returned result:

```

Summary:
resource cost: cpu 1.03 Core * Min, memory 1.99 GB * Min
inputs:
  meta_dev.poi_sample_wkt: 1000 (237592 bytes)
  meta_dev.link_sample_wkt: 1000 (105940 bytes)
outputs:
Job run time: 41.000
+-----+-----+
| link_id | shape_id |
+-----+-----+
| 5121371185457659960 | B000A844XK |
| 5121377123249946651 | B000A85TV4 |
| 5121377166199619654 | B000A844KT |
+-----+-----+

```

6.12.9.9. ST_KNNFromRTree

This topic describes the ST_KNNFromRTree function and provides an example of using this function.

Function declaration:

```
ST_KNNFromRTree(string uniqueId, string geometryWkt, int k, RTree rtree)
```

Description: This function is a user-defined table-valued function (UDTF). It uses the unique ID and well-known text (WKT) string of each geometry and the R-tree index that is created by calling ST_BuildRTreeIndex as input parameters. This function returns the IDs of k objects that are near to the geometry from the R-tree index.

Example:

Create an R-tree for all points in a table and use the KNN function to find the nearest point of each point.

```

SELECT /*+mapjoin(i)*/
  geospatial.ST_KNNFromRTree(id, point, 1, i.index) AS (id1, id2)
FROM poi_sample_wkt
JOIN
(
  SELECT geospatial.ST_BuildRTreeIndex(id, point) AS index
  FROM poi_sample_wkt
) i;

```

Returned result:

```

Summary:
resource cost: cpu 1.17 Core * Min, memory 2.24 GB * Min
inputs:
  meta_dev.poi_sample_wkt: 1000 (237592 bytes)
outputs:
Job run time: 46.000
+-----+-----+
| id1 | id2 |
+-----+-----+
| B000A01B4E | B000A01B4E |
| B000A01C19 | B000A01C19 |
| B000A023A5 | B000A023A5 |
| B000A02F81 | B000A02F81 |
| B000A07BEE | B000A07BEE |
| B000A07E06 | B000A07E06 |
| B000A08863 | B000A08863 |
...
-- The table has 1,000 rows of data. This function returns 1,000 rows of data, which meets your expectations.

```

6.12.10. Other functions

6.12.10.1. ST_IsValid

This topic describes the ST_IsValid function and provides an example of using this function.

Function declaration:

```

boolean ST_IsValid(st_geometry geometry)
boolean ST_IsValid(string wkt)

```

Description: This function checks whether the input geometry or well-known text (WKT) string meets the requirements.

Example:

```

SELECT ST_IsValid('POINT(-102.849854 36.451113)');
SELECT ST_IsValid(ST_Point('POINT(-102.849854 36.451113)'));

```

6.12.10.2. ST_Transform

This topic describes the ST_Transform function and provides an example of using this function.

Function declaration:

```

st_geometry ST_TransformWGS84(st_geometry geometry)
st_geometry ST_Transform(st_geometry geometry, integer toSRID)
st_geometry ST_Transform(st_geometry geometry, integer fromSRID, integer toSRID)

```

Description: This function converts the coordinates of the input geometry from one spatial reference system to another. The ST_TransformWGS84 function converts the coordinates of the geometry from EPSG:4326 to EPSG:3857. The ST_Transform function converts the geometry from fromSRID to toSRID. If the overload function contains only toSRID, you must call the ST_SetSRID function first.

Example:

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450, 743265 2967450,743265.625 2967416,743238 2967416)'), 2249, 4326)));
SELECT ST_AsText(ST_TransformWGS84(ST_GeomFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009, -71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 42.3902896512902)'))));
```

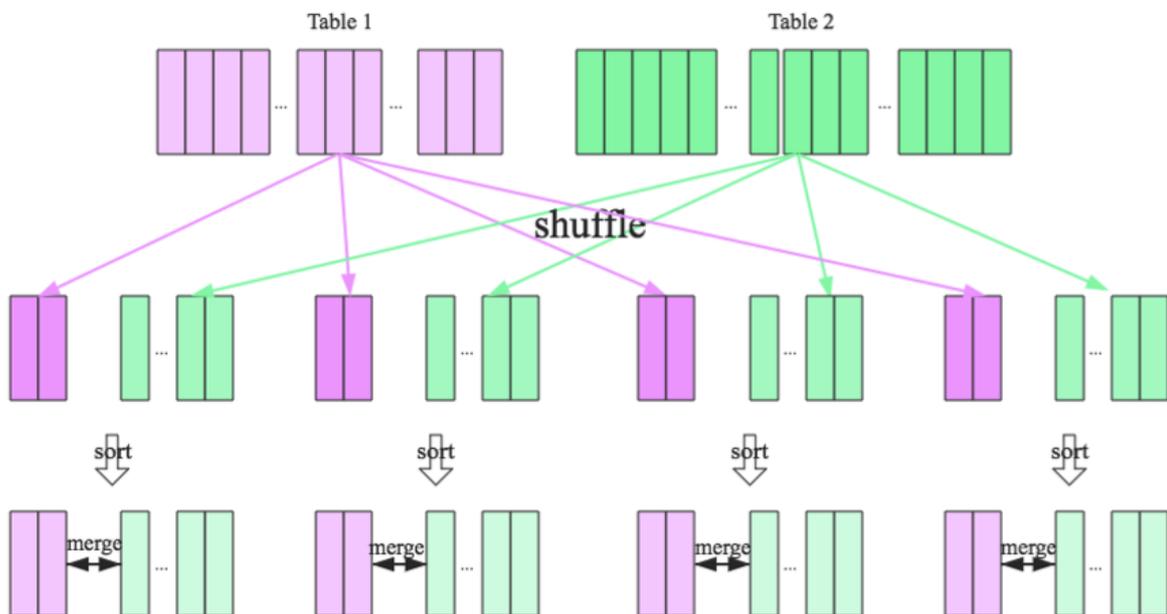
6.13. MaxCompute Hash Clustering

6.13.1. Background information

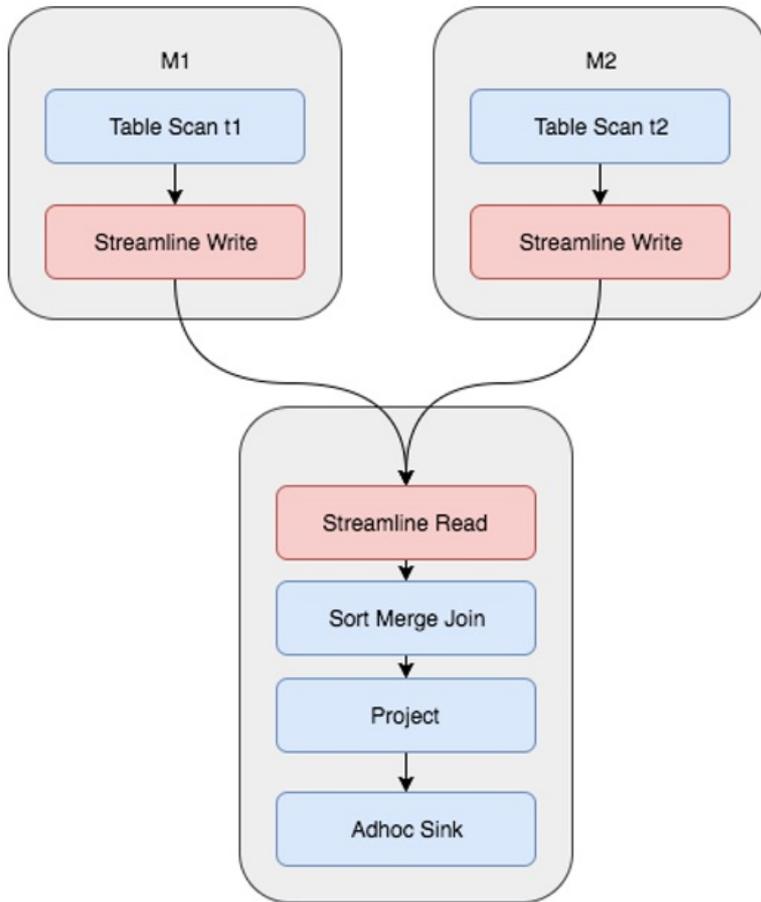
This topic describes the background of MaxCompute Hash Clustering.

JOIN operations are commonly used for queries in MaxCompute. MaxCompute provides the following methods to implement JOIN operations:

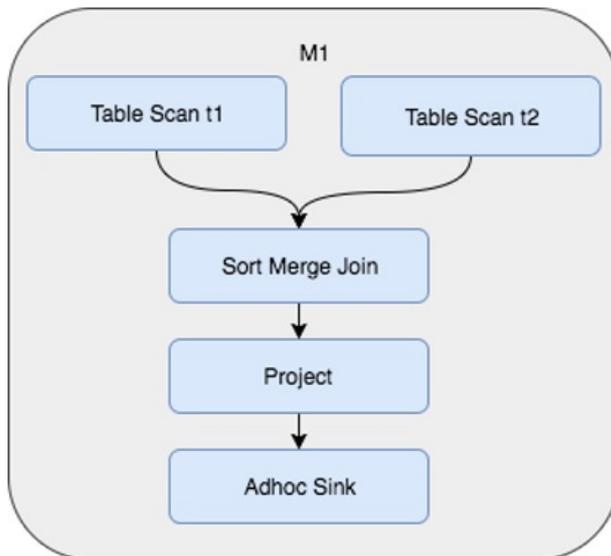
- Broadcast hash join: This method is used when a JOIN operation involves a small table. The small table is broadcasted and transferred to all instances in a join task. Then, the hash join operation is performed to join the small table with a large table.
- Shuffle hash join: This method is used when a JOIN operation involves large tables that cannot be directly broadcasted. In this case, the hash shuffle operation is performed on two tables based on join keys. The hash results for the same key-value pairs are the same. This ensures that results that have the same key are collected to the same instance of a join task. For each instance, a hash table is created by using a small table, probe operations are performed by using a large table, and then the tables are joined.
- Sort merge join: This method is used when a JOIN operation involves larger tables and the preceding methods cannot be used because the memory is insufficient to create a hash table. In this case, the hash shuffle operation is performed on two tables based on join keys, the obtained values are sorted by using join keys, and then the sorted values are merged.



The sort merge join method is commonly used in MaxCompute because MaxCompute processes large amounts of data in most cases. If you use this method, shuffle and join operations are repeatedly performed. The physical execution plan of Job Scheduler of a JOIN operation requires multiple stages, which consumes excessive amounts of resources.



To resolve this issue, MaxCompute allows you to configure the hash shuffle and sort attributes when the data is initially generated in a table. This prevents data from being repeatedly shuffled and sorted in subsequent queries. The number of stages in the physical execution plan of Job Scheduler of a JOIN operation is also reduced. The preceding figure shows that only one stage is required.



MaxCompute Hash Clustering allows you to configure the shuffle and sort attributes of a table when you create the table. Then, MaxCompute optimizes the execution plan, improves the efficiency, and saves resources based on the existing storage characteristics.

6.13.2. Descriptions

6.13.2.1. Enable Hash Clustering

This topic describes how to enable the Hash Clustering feature of MaxCompute.

The Hash Clustering feature has been launched. By default, this feature is enabled. If you want to use clustered indexes, add the following flag:

```
set odps.sql.cfile2.enable.read.write.index.flag=true;
```

After the flag is set to true, the system automatically creates indexes for the sorted hash buckets to improve query efficiency. To use clustered indexes, you must add this flag during table creation and subsequent queries. If you want to use clustered indexes in your project at all times, contact the MaxCompute team.

Note Clustered indexes improve the efficiency of queries (equivalent values or ranges) based on sort keys. You can experience the superior performance provided by Hash Clustering even if you do not add this flag.

6.13.2.2. Create a hash-clustered table

This topic describes how to create a hash-clustered table.

You can use the following statement to create a hash-clustered table. You must specify cluster keys or hash keys and the number of hash buckets. The sort operation is optional. However, we recommend that you use the same parameter values as the cluster keys to achieve optimal performance.

```
CREATE TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [comment col_comment], ...)]
  [comment table_comment]
  [PARTITIONED BY (col_name data_type [comment col_comment], ...)]
  [CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC | DESC]
  ] ...))] INTO number_of_buckets BUCKETS]
  [AS select_statement]
```

You can use the following statement to create a standard table:

```
CREATE TABLE T1 (a string, b string, c bigint) CLUSTERED BY (c) SORTED BY (c) INTO 1024 BUCKETS;
```

You can use the following statement to create a partitioned table:

```
CREATE TABLE T1 (a string, b string, c bigint) PARTITIONED BY (dt string) CLUSTERED BY (c) SORTED BY
(c) INTO 1024 BUCKETS;
```

The following sections detail the CLUSTERED BY, SORTED BY, and INTO number_of_buckets BUCKETS clauses.

CLUSTERED BY

The CLUSTERED BY clause specifies hash keys. MaxCompute performs the hash operation on the specified column and distributes data to buckets based on the hash values. To prevent data skew and hot spots, and to concurrently execute statements, we recommend that you specify a column that has large value ranges and a small number of duplicate key-value pairs in CLUSTERED BY. In addition, to optimize the JOIN operation, we recommend that you select commonly used join or aggregation keys. The join and aggregation keys are similar to the primary keys in conventional databases.

SORTED BY

The SORTED BY clause specifies how fields are sorted in a bucket. We recommend that you specify the same column in SORTED BY as that in CLUSTERED BY to improve execution efficiency. After you specify the column in SORTED BY, MaxCompute automatically generates indexes and then executes SQL statements faster when you query data based on these indexes.

INTO number_of_buckets BUCKETS

The INTO number_of_buckets BUCKETS clause specifies the number of hash buckets, which is required. The number of hash buckets is determined by the volume of data. More buckets indicate higher concurrency, which shortens the job running time. However, if a large number of buckets exist, excessive small files may be generated. In addition, high concurrency increases CPU time. We recommend that you set the volume of data for each bucket to a value that ranges from 500 MB to 1 GB. If a large table is used, you can adjust the value to a larger value as required.

You can remove the shuffle operation only for tables with the same number of buckets in MaxCompute. In later versions, MaxCompute will support bucket alignment. You will be able to remove the shuffle operation for tables whose numbers of buckets are multiples or factors of each other. To achieve bucket alignment, we recommend that you set the number of buckets to a power of 2, for example, 512, 1,024, and 2,048. The maximum number of buckets is 4,096. If the number of buckets exceeds the value, the performance and resource usage may be affected.

If you want to remove the shuffle and sort operations during a JOIN operation on two tables, the numbers of hash buckets in the tables must be the same. If the numbers that are calculated based on the aforementioned method are inconsistent, we recommend that you use the larger number for the JOIN operation. This ensures that SQL statements can be concurrently executed in an efficient manner.

If the sizes of two tables greatly differ, you can set the number of buckets for the large table to several times of that for the small table, for example, 256 and 1,024. If automatic hash bucket split and merging are supported, the settings can be optimized by using data features.

6.13.2.3. Modify table attributes

This topic describes how to modify the attributes of a hash-clustered table.

For a partitioned table, MaxCompute allows you to execute the ALTER TABLE statement to add the Hash Clustering attribute to a table or remove the Hash Clustering attribute from a table.

```
ALTER TABLE table_name
    [CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC |
DESC] ...])] INTO number_of_buckets BUCKETS]
ALTER TABLE table_name NOT CLUSTERED;
```

When you use the ALTER TABLE statement, take note of the following points:

- The ALTER TABLE statement can modify only the Hash Clustering attribute of a partitioned table. The Hash Clustering attribute cannot be modified after it is added to a non-partitioned table.
- The ALTER TABLE statement takes effect only on the new partitions of a table, which include the partitions generated by using the INSERT OVERWRITE statement. New partitions are stored based on the Hash Clustering attribute. The storage formats of existing partitions remain unchanged.
- The ALTER TABLE statement takes effect only on the new partitions of a table. Therefore, you cannot specify a partition in this statement.

The ALTER TABLE statement is suitable for existing tables. After the Hash Clustering attribute is added, new partitions are stored based on the Hash Clustering attribute.

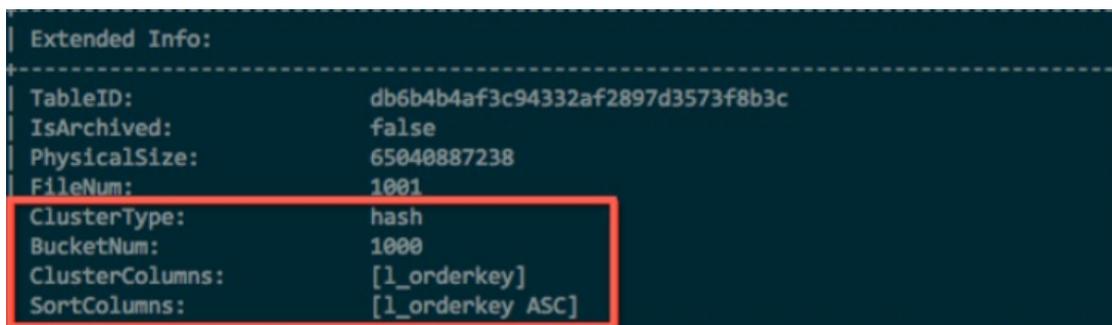
6.13.2.4. View and verify table attributes

This topic describes how to view and verify attributes of a hash-clustered table.

After you create a hash-clustered table, execute the following statement to view table attributes:

```
DESC EXTENDED table_name;
```

The table attributes are displayed in Extended Info.



```
Extended Info:
-----
TableID:          db6b4b4af3c94332af2897d3573f8b3c
IsArchived:       false
PhysicalSize:     65040887238
FileNum:          1001
ClusterType:      hash
BucketNum:        1000
ClusterColumns:   [1_orderkey]
SortColumns:      [1_orderkey ASC]
```

You can also execute the following statement to view partition attributes of a partitioned table:

```
DESC EXTENDED table_name partition(pt_spec);
```

The following figure shows the execution result.

```

PartitionSize: 754
-----
CreateTime:      2017-07-07 14:01:03
LastDDLTime:    2017-07-07 14:01:03
LastModifiedTime: 2017-07-07 14:01:03
-----
IsExstore:      false
IsArchived:     false
PhysicalSize:   2262
FileNum:        2
ClusterType:    hash
BucketNum:      500
ClusterColumns: [c1]
SortColumns:    [c1 ASC]

```

6.13.3. Benefits

6.13.3.1. Bucket pruning and index optimization

This topic describes bucket pruning and index optimization.

The following code provides a syntax sample:

```

CREATE TABLE t1 (id bigint, a string, b string) CLUSTERED BY (id) SORTED BY (id) INTO 1000 BUCKETS;
...
SELECT t1.a, t1.b, t1.c FROM t1 WHERE t1.id=12345;

```

This syntax indicates a full scan for a standard table. A full scan for a large table consumes a large number of resources. However, if the hash shuffle operation is performed on all id fields and the id fields are sorted, the query is greatly simplified.

Sample procedure:

1. Find the hash bucket that corresponds to 12345. This query is performed in only one bucket, not all 1,000 buckets. This process is called bucket pruning.
2. Data in a bucket is stored based on IDs. MaxCompute automatically creates indexes and uses the INDEX LOOKUP function to locate relevant records.

The simplified procedure not only greatly reduces the number of mappers, but also allows mappers to locate the page where the data is stored by using the INDEX function. Therefore, the volume of loaded data is greatly reduced.

6.13.3.2. Aggregation optimization

This topic describes aggregation optimization.

The following code provides an example:

```

SELECT department, SUM(salary) FROM employee GROUP BY (department);

```

In most cases, the department column is shuffled and sorted. Then, a stream aggregate operation is performed to collect statistics on the department groups. However, if `CLUSTERED BY (department) SORTED BY (department)` is executed for the table data, the shuffle and sort operations are no longer required.

6.13.3.3. Storage optimization

This topic describes storage optimization.

In addition to computation optimization, storage space is greatly saved if tables are shuffled and stored in a sorted manner. MaxCompute uses column store at the underlying layer. Records that have the same or similar key-value pairs are stored together by the sort function, which facilitates encoding and compression. This way, compression efficiency is improved. In some extreme cases, a sorted table can save 50% more storage space than an unsorted table. Therefore, Hash Clustering is suitable for storing tables that have long lifecycles.

The following figure shows a sample table with 100 GB of TPC-H line items and multiple data types, such as INT, DOUBLE, and STRING. If Hash Clustering is used, about 10% of the storage space is saved when the volume of data and compression format remain unchanged.

```
CreateTime:          2016-04-17 21:48:08
LastDDLTime:        2016-04-17 21:48:08
LastModifiedTime:   2016-04-17 21:50:10
-----
InternalTable: YES  | Size: 23573055432
-----
Native Columns:
-----
Field      | Type      | Label | Comment
-----
l_orderkey | bigint    |       |
l_partkey  | bigint    |       |
l_suppkey  | bigint    |       |
l_linenum  | bigint    |       |
l_quantity | double    |       |
l_extendedprice | double  |       |
l_discount | double    |       |
l_tax      | double    |       |
l_returnflag | string   |       |
l_linestatus | string   |       |
l_shipdate | string    |       |
l_commitdate | string   |       |
l_receiptdate | string  |       |
l_shipinstruct | string  |       |
l_shipmode | string    |       |
l_comment  | string    |       |
```

```

| CreateTime:          2017-07-13 14:40:11
| LastDDLTime:        2017-07-13 14:40:11
| LastModifiedTime:   2017-07-13 15:05:04
+-----+-----+
| InternalTable: YES  | Size: 21658913950
+-----+-----+
| Native Columns:
+-----+-----+
| Field      | Type   | Label | Comment
+-----+-----+
| l_orderkey | bigint |       |
| l_partkey  | bigint |       |
| l_suppkey  | bigint |       |
| l_linenumb | bigint |       |
| l_quantity | double |       |
| l_extended | double |       |
| l_discount | double |       |
| l_tax      | double |       |
| l_returnfl | string |       |
| l_linestat | string |       |
| l_shipdat | string |       |
| l_commitda | string |       |
| l_receiptd | string |       |
| l_shipinsh | string |       |
| l_shipmod | string |       |
| l_comment  | string |       |

```

6.13.4. ShuffleRemove

This topic describes the operations that are related to ShuffleRemove.

- Range-clustered tables support the join and aggregate operations. If a join key or group key is a range-clustered key or its prefix, data redistribution is not required. This mechanism is called ShuffleRemove, which improves execution efficiency.

Usage: The `odps.optimizer.enable.range.partial.repartitioning` flag specifies whether to enable this feature. By default, this feature is disabled.

- If the two hash-clustered tables you want to join have different numbers of buckets but the numbers are multiples of each other, data redistribution is not required. This improves execution efficiency.

Usage: The `odps.optimizer.enable.hash.partial.repartitioning` flag specifies whether to enable this feature. By default, this feature is enabled.

- Correlated Shuffle Remove is supported. If data meets distribution requirements but does not meet the sorting requirements, you can add a sort operator to avoid data redistribution.

6.13.5. Limits

This topic describes limits on MaxCompute Hash Clustering.

Limits:

- The INSERT INTO statement is not supported. You can only execute the INSERT OVERWRITE statement to add data.
- Small files cannot be merged. Most data is evenly distributed in buckets when it is split. Therefore, the number of small files is not large. If you merge files, the data distribution is affected. However, you can still use the merge and archive commands to change the storage format of a table file and the format of a RAID file.

- You cannot use Tunnel to upload data to a range-clustered table because the data uploaded by using Tunnel is unsorted.

6.14. Common MaxCompute SQL parameter settings

6.14.1. Map configurations

This topic describes Map configurations.

```
set odps.sql.mapper.cpu=100;
```

Description: specifies the number of CPUs used by each instance in a Map task. Default value: 100. Valid values: [50,800].

```
set odps.sql.mapper.memory=1024;
```

Description: specifies the memory size of each instance in a Map task. Unit: MB. Default value: 1024. Valid values: [256,12288].

```
set odps.sql.mapper.merge.limit.size=64;
```

Description: specifies the maximum size of control files to be merged. Unit: MB. Default value: 64. Valid values: [0,Integer.MAX_VALUE]. You can configure this variable to control the input of mappers.

```
set odps.sql.mapper.split.size=256;
```

Description: specifies the maximum data input volume for a Map task. Unit: MB. Default value: 256. Valid values: [1,Integer.MAX_VALUE]. You can configure this variable to control the input of mappers.

6.14.2. Join configurations

This topic describes Join configurations.

```
set odps.sql.joiner.instances=-1;
```

Description: specifies the number of instances of a join task. Default value: -1. Valid values: [0,2000].

```
set odps.sql.joiner.cpu=100;
```

Description: specifies the number of CPUs used by each instance of a join task. Default value: 100. Valid values: [50,800].

```
set odps.sql.joiner.memory=1024;
```

Description: specifies the memory size of each instance of a join task. Unit: MB. Default value: 1024. Valid values: [256,12288].

6.14.3. Reduce configurations

This topic describes Reduce configurations.

```
set odps.sql.reducer.instances=-1;
```

Description: specifies the number of instances in a Reduce task. Default value: -1. Valid values: [0,2000].

```
set odps.sql.reducer.cpu=100;
```

Description: specifies the number of CPUs used by each instance in a Reduce task. Default value: 100. Valid values: [50,800].

```
set odps.sql.reducer.memory=1024;
```

Description: specifies the memory size of each instance in a Reduce task. Unit: MB. Default value: 1024. Valid values: [256,12288].

6.14.4. UDF configurations

This topic describes the configurations of user-defined functions (UDFs).

```
set odps.sql.udf.jvm.memory=1024;
```

Description: specifies the maximum memory size used by the UDF JVM heap. Unit: MB. Default value: 1024. Valid values: [256,12288].

```
set odps.sql.udf.timeout=600;
```

Description: specifies the timeout period of a UDF. Unit: seconds. Default value: 600. Valid values: [0,3600].

```
set odps.sql.udf.python.memory=256;
```

Description: specifies the maximum memory size used by the UDF Python API. Unit: MB. Default value: 256. Valid values: [64,3072].

```
set odps.sql.udf.optimize.reuse=true/false;
```

Description: When this parameter is set to true, each UDF function expression can be calculated only once. This improves performance. Default value: true.

```
set odps.sql.udf.strict.mode=false/true;
```

Description: specifies whether functions return NULL or an error if dirty data is found. If the parameter is set to true, an error is returned. Otherwise, NULL is returned.

6.14.5. MapJoin configurations

This topic describes MapJoin configurations.

```
set odps.sql.mapjoin.memory.max=512;
```

Description: specifies the maximum memory size for a small table that is involved in a MapJoin operation. Unit: MB. Default value: 512. Valid values: [128,2048].

```
set odps.sql.reshuffle.dynamicpt=true/false;
```

Description:

- Dynamic partitioning is time-consuming in some scenarios. Disabling dynamic partitioning can accelerate SQL.
- If the number of dynamic partitions is small, disabling dynamic partitioning can prevent data skew.

6.14.6. Configurations of data skew

This topic describes configurations of data skew.

```
set odps.sql.groupby.skewindata=true/false;
```

Description: allows you to enable GROUP BY optimization.

```
set odps.sql.skewjoin=true/false;;
```

Description: allows you to enable JOIN optimization. This is effective only when `odps.sql.skewinfo` is specified.

```
set odps.sql.skewinfo;
```

Description: allows you to configure detailed information for JOIN optimization. Command syntax:

```
set odps.sql.skewinfo=skewed_src:(skewed_key) [ ("skewed_value") ]
```

Example:

Configure a single skewed value for a single field.

```
set odps.sql.skewinfo=src_skewjoin1:(key) [ ("0") ];  
-- Output result: explain select a.key c1, a.value c2, b.key c3, b.value c4 from src a join src_skew  
join1 b on a.key = b.key;
```

Configure multiple skewed values for a single field.

```
set odps.sql.skewinfo=src_skewjoin1:(key) [ ("0") ("1") ];  
-- Output result: explain select a.key c1, a.value c2, b.key c3, b.value c4 from src a join src_skew  
join1 b on a.key = b.key;
```

6.15. MapReduce-to-SQL conversion for execution

6.15.1. Overview

This topic provides an overview of the MapReduce-to-SQL conversion for execution feature.

MaxCompute provides a series of Java APIs for MapReduce to process data.

MapReduce programs are automatically converted to SQL for execution. After the conversion, you can use the compiler, cost-based optimizer, and vectorized execution engine released with MaxCompute V2.0 to process the MapReduce programs. The new features of the SQL engine can also be used. The features, performance, and stability of the SQL engine are optimized.

Note

- You do not need to change the original APIs and job logic.
- Only MapReduce jobs of the OpenMR type, which are written by using MapReduce APIs, can be converted to SQL.
- This feature can be used only for projects and jobs.
- This feature supports views as the input.
- This feature supports external tables as the input.
- This feature supports TemporaryFile reads and writes.
- This feature allows you to read data from and write data to hash-clustered tables.
- This feature supports near-real-time execution of small jobs.

6.15.2. Configure the MaxCompute client

This topic describes how to configure the MaxCompute client.

1. Download the latest **MaxCompute client** package to your computer and configure the client.

Note On the download page, you can obtain the MaxCompute client package of the required version in the **Other Resources** section.

2. Configure the execution mode.

You can configure the execution mode based on your business requirements. The default execution mode is lot. In lot mode, jobs are run by MapReduce. The new compiler, optimizer, and execution engine are not used.

You can enable the execution mode by setting the `odps.mr.run.mode` parameter. Valid values: lot, sql, and hybrid.

- Method 1: Enable the execution mode at the project level. In this case, the configuration takes effect on all jobs in the project. Therefore, only the project administrator can apply to enable the execution mode. Set the `odps.mr.run.mode` parameter to hybrid or sql. If SQL execution fails in hybrid mode, the job is run by MapReduce. If SQL execution fails in sql mode, an error is reported.
- Method 2: Enable the execution mode at the session level. This method applies only to the current job. To enable the execution mode, use one of the following methods:
 - Add a set flag, such as `set odps.mr.run.mode=hybrid;` before JAR statements.
 - Configure job parameters. Example:

```
JobConf job = new JobConf();
job.set("odps.mr.run.mode", "hybrid")
```

MaxCompute O&M personnel can enable the execution mode at the project level at a later point in time.

6.15.3. Configure running settings in DataWorks

This topic describes how to configure running settings in DataWorks.

Jobs that run in DataWorks are updated by the O&M personnel of MaxCompute and DataWorks. You do not need to update your client.

- Enable the conversion for a single job.

You can add a set flag before the statement of a MapReduce job or configure job parameters for the set flag. These methods take effect at the session level and apply only to the current job.

The following examples show how to use these methods:

- Add a set flag, such as `set odps.mr.run.mode=hybrid;` before JAR statements.
- Configure job parameters. Example:

```
JobConf job = new JobConf();
job.set("odps.mr.run.mode","hybrid")
```

- Enable the conversion at the project level by setting `odps.mr.run.mode` for a project.

6.15.4. View details

This topic describes how to view MapReduce-to-SQL conversion results and running details of SQL jobs.

You can use Logview and MaxCompute Studio to view MapReduce-to-SQL conversion results and running details of SQL jobs.

- Logview XML

Open Logview and click the LOT node in the center of the page. The SQL jobs that are converted from MapReduce jobs are included in the XML information about the node. Example:

```
create temporary function mr2sql_mapper_152955927079392291755 as 'com.aliyun.odps.mapred.bridge.LotMapperUDTF' using ;
create temporary function mr2sql_reducer_152955927079392291755 as 'com.aliyun.odps.mapred.bridge.LotReducerUDTF' using ;
@sub_query_mapper :=
SELECT k_id,v_gmt_create,v_gmt_modified,v_product_id,v_admin_seq,v_sku_attr,v_sku_price,v_sku_stock,v_sku_code,v_sku_image,v_delivery_time,v_sku_bulk_order,v_sku_bulk_discount,v_sku_image_version,v_currency_code
FROM(
SELECT mr2sql_mapper_152955927079392291755(id,gmt_create,gmt_modified,product_id,admin_seq,sku_attr,sku_price,sku_stock,sku_code,sku_image,delivery_time,sku_bulk_order,sku_bulk_discount,sku_image_version,currency_code) as (k_id,v_gmt_create,v_gmt_modified,v_product_id,v_admin_seq,v_sku_attr,v_sku_price,v_sku_stock,v_sku_code,v_sku_image,v_delivery_time,v_sku_bulk_order,v_sku_bulk_discount,v_sku_image_version,v_currency_code)
FROM ae_antispam.product_sku_tt_inc
WHERE ds = "20180615" AND hh = "21"
UNION ALL
SELECT mr2sql_mapper_152955927079392291755(id,gmt_create,gmt_modified,product_id,admin_seq,sku_attr,sku_price,sku_stock,sku_code,sku_image,delivery_time,sku_bulk_order,sku_bulk_discount,sku_image_version,currency_code) as (k_id,v_gmt_create,v_gmt_modified,v_product_id,v_admin_seq,v_sku_attr,v_sku_price,v_sku_stock,v_sku_code,v_sku_image,v_delivery_time,v_sku_bulk_order,v_sku_bulk_discount,v_sku_image_version,v_currency_code)
FROM ae_antispam.product_sku
) open_mr_alias1
DISTRIBUTE BY k_id SORT BY k_id ASC;
@sub_query_reducer :=
SELECT mr2sql_reducer_152955927079392291755(k_id,v_gmt_create,v_gmt_modified,v_product_id,v_admin_seq,v_sku_attr,v_sku_price,v_sku_stock,v_sku_code,v_sku_image,v_delivery_time,v_sku_bulk_order,v_sku_bulk_discount,v_sku_image_version,v_currency_code) as (id,gmt_create,gmt_modified,product_id,admin_seq,sku_attr,sku_price,sku_stock,sku_code,sku_image,delivery_time,sku_bulk_order,sku_bulk_discount,sku_image_version,currency_code)
FROM @sub_query_mapper;
FROM @sub_query_reducer
INSERT OVERWRITE TABLE ae_antispam.product_sku
SELECT id,gmt_create,gmt_modified,product_id,admin_seq,sku_attr,sku_price,sku_stock,sku_code,sku_image,delivery_time,sku_bulk_order,sku_bulk_discount,sku_image_version,currency_code ;
```

- Logview details or summary

A new execution engine is used to run jobs.

```
Job run mode: fuxi job
Job run engine: execution engine
```

- Logview details or JSON summary

The JSON summary information in MapReduce contains only the input and output information about Map and Reduce. However, the JSON summary information in SQL allows you to view details about each stage of SQL execution, such as all execution parameters, logical execution plans, physical execution plans, and execution details. Example:

```
"midlots" :
[
  "LogicalTableSink(table=[[odps_flighting.flt_20180621104445_step1_ad_quality_tech_qp_algo_antifake_wordbag_filter_bag_change_result_lv2_20, auctionid,word,match_word(3) {0, 1, 2}]]
  OdpsLogicalProject(auctionid=[${0}], word=[${1}], match_word=[${2}])
  OdpsLogicalProject(auctionid=[${0}], word=[${1}], match_word=[${2}])
  OdpsLogicalProject(auctionid=[${0}], word=[${1}], match_word=[${2}])
  OdpsLogicalProject(auctionid=[${2}], word=[${3}], match_word=[${4}])
  OdpsLogicalTableFunctionScan(invocation=[[MR2SQL_MAPPER_152955294118813063732 (${0}, ${1})]()], rowType
  =[RecordType(VARCHAR(2147483647) item_id, VARCHAR(2147483647) text, VARCHAR(2147483647) __tf_0_0,
  VARCHAR(2147483647) __tf_0_1, VARCHAR(2147483647) __tf_0_2)])
  OdpsLogicalTableScan(table=[[ad_quality_tech_qp_algo_antifake_wordbag_filter_bag_change_lv2_20, item_id,text(2) {0, 1}]]
  ]
```

6.15.5. Perform operations on the distributed file system

This topic describes how to perform operations on the distributed file system.

Procedure

1. Specify volume files.

You can use one of the following methods to specify volume files:

- Use a utility class to specify the input and output files:

```
com.aliyun.odps.mapred.utils.InputUtils.addVolume( new VolumeInfo([project,]inVolume,inPartition, "inLabel"), new JobConf());
com.aliyun.odps.mapred.utils.OutputUtils.addVolume( new VolumeInfo([project,]outVolume, outPartition, "outLabel"), new JobConf());
```

In the preceding commands, project and label are optional, and the current project and default label are used by default. If multiple input and output files are used, labels are used to distinguish the files from each other. Authorization is required before you access the volume files of other projects.

- Configure parameters to specify the volume and partition of the input and output files. If multiple input or output files are used, separate the parameters with commas (,).

```
set odps.sql.volume.input[/output].desc = [<project>.<table>.<partition>[:<label>];
```

2. Call the following method by using a context object in the map and reduce steps to write data to the distributed file system or write data stream input and output files:

```
context.getOutputStreamVolumeFileSystem();
```

6.16. Analysis of the mapping between SQL input and output fields

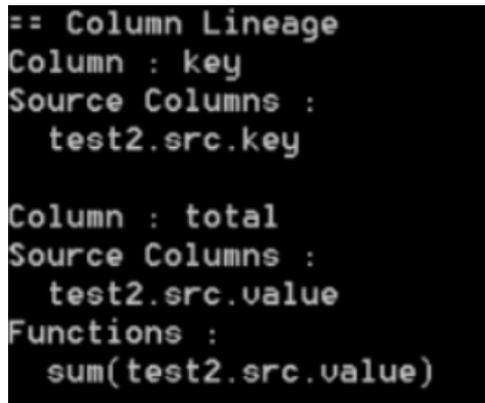
6.16.1. Overview

This topic describes the feature of analyzing the mapping between fields in input and output tables.

Example:

```
select key, sum(value) as total from src group by key;
```

The result that is shown in the following figure is returned.



```
= Column Lineage
Column : key
Source Columns :
  test2.src.key

Column : total
Source Columns :
  test2.src.value
Functions :
  sum(test2.src.value)
```

Two columns are returned: key and total. The key column corresponds to the src.key column of the input table. The total column corresponds to the src.value column of the input table.

6.16.2. Usage notes

This topic provides an example on how to use the feature of analyzing the mapping between fields in input and output tables.

Output format

Field mapping analysis supports human-readable and JSON formats. You can use the `set odps.sql.select.output.format=HumanReadable/json` flag to specify the output format.

SDK-based field mapping analysis

Example

```
Odps odps = initOdps();
// To perform analysis, use LineageTask.
LineageTask task = new LineageTask("task_name", "select * from dual;");
// Optional. Use the preceding flag to specify the output format.
Map<String, String> settings = new LinkedHashMap<>();
settings.putIfAbsent("odps.sql.select.output.format", "json");
task.setProperty("settings", JSON.toJSONString(settings));
// Submit code to the server for field mapping analysis.
Instance instance = odps.instances().create(task);
System.out.println(instance.getId());
String logView = odps.logview().generateLogView(instance, 72);
System.out.println(logView);
instance.waitForSuccess();
// Obtain the analysis result.
System.out.println(instance.getTaskResults().get("task_name"));
```

odpscmd-based field mapping analysis

Examples

CLI mode: Use the `-X` parameter for field mapping analysis.

```
./bin/odpscmd.bat -X D:\lineage.q
```

Interactive mode: After you enter the interactive mode of `odpscmd`, you can use the preceding flag to specify the output format. The usage method is similar to that used to submit SQL jobs.

```
lineage_test>set odps.sql.task.mode=LINEAGE;
-- OK is returned. Continue to run the following command:
odps@ lineage_test>set odps.sql.select.output.format=Humanreadable;
-- OK is returned. Continue to run the following command:
lineage_test>select * from dual;
```

Returned result:

```
== Column Lineage
Column : id
Source Columns :
      test2.dual.id
```

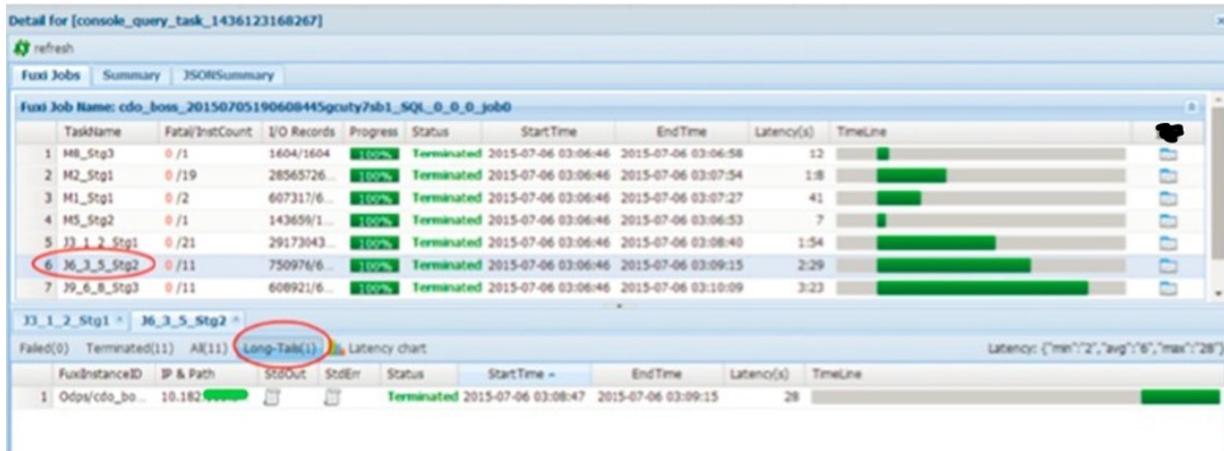
6.17. Common MaxCompute SQL errors and solutions

6.17.1. Data skew

6.17.1.1. Overview

This topic provides an overview of data skew issues.

In most cases, data skew occurs if the min, max, and avg values of the instance time, input records, and output records parameters are unbalanced in an instance on which jobs run. For example, if the max value is significantly larger than the avg value, data skew may occur. You can use Logview to identify data skew issues, as shown in the following figure.



The Long Tails tab of each task displays the instances on which data skew occurs. The root cause of data skew is that the amount of data processed by a specific instance far exceeds that processed by other instances. As a result, the running time of this instance is also much longer than the average running time of other instances, and the entire job slows down.

Different types of data skew issues in SQL are resolved by using different methods. The subsequent topics describe possible causes of data skew issues and provide solutions.

6.17.1.2. GROUP BY data skew

This topic describes possible causes of the GROUP BY data skew and provides a solution.

Possible cause: GROUP BY keys are unevenly distributed, which results in data skew on reducers.

Solution: Set the anti-skew parameter before you execute SQL statements.

```
set odps.sql.groupby.skewindata=true;
```

Note If you set this parameter to true, when the Shuffle hash algorithm is used, the system automatically adds a random factor and introduces a new task to prevent data skew.

6.17.1.3. DISTRIBUTE BY data skew

This topic describes possible causes of the DISTRIBUTE BY data skew and provides a solution.

Possible cause: Constants are used to execute the DISTRIBUTE BY clause for full sorting of the entire table. This results in data skew on reducers.

Solution: Avoid the preceding operation.

6.17.1.4. Data skew caused by JOIN operations

This topic describes the data skew caused by JOIN operations and provides a solution.

Possible cause: JOIN ON keys are unevenly distributed. For example, a key has a large number of duplicates in multiple tables that you want to join. This results in a Cartesian explosion of data in the instance on which the JOIN statement is executed and causes data skew.

Solution: You can use one of the following methods to resolve the data skew issue based on scenarios:

- If one of the tables that you want to join is a small table, execute the MAP JOIN statement, instead of the JOIN statement.
- Use separate logic to handle skewed keys. For example, if data skew occurs because a large number of null key values exist in both tables, you must filter out these null key values or use a CASE WHEN expression to replace the null key values with random values before you perform a JOIN operation.
- If you do not want to change the SQL statement, configure the following parameters to enable automatic optimization on MaxCompute:

```
set odps.sql.skewinfo=tab1:(col1,col2)[(v1,v2),(v3,v4),...];
set odps.sql.skewjoin=true;
```

6.17.1.5. Data skew caused by multiple DISTINCT operations

This topic describes the data skew caused by multiple DISTINCT operations and provides a solution.

Possible cause: Multiple DISTINCT operations aggravate the GROUP BY skew.

Solution: Execute two GROUP BY clauses to alleviate the data skew issue. In most cases, do not use multiple DISTINCT operations at the same time.

6.17.1.6. Data skew caused by misuse of dynamic partitions

This topic describes the data skew caused by misuse of dynamic partitions and provides a solution.

Possible cause: In scenarios where dynamic partitions are used, if K Map instances and N destination partitions are deployed, $K \times N$ small files may be generated. This significantly increases the management workload of the file system. Therefore, the following configuration takes effect by default:

```
set odps.sql.reshuffle.dynamicpt=true;
```

An additional level of Reduce task is introduced. The same or a small number of other Reduce instances are used to write data to the same destination partition. This prevents a large number of small files from being generated. However, dynamic partition shuffle may cause data skew.

Solution: If the number of destination partitions is small, only an appropriate number of small files are generated. In this case, you can run the following command to set odps.sql.reshuffle.dynamicpt to false to prevent the data skew caused by dynamic partition shuffle.

```
set odps.sql.reshuffle.dynamicpt=false;
```

If the number of destination partitions is large, we recommend that you do not use dynamic partitions because a large number of small files are generated.

6.17.1.7. Use SKEWJOIN HINT to avoid skewed hot key values

This topic describes how to avoid skewed hot key values by using SKEWJOIN HINT. It also provides examples for reference.

Methods

- Method 1: Specify the name of a table in SKEWJOIN HINT. Note that the name specified in the hint is the alias of the table. The following statement shows an example:

```
select /*+ skewjoin(a) */ * from T0 a join T1 b on a.c0 = b.c0 and a.c1 = b.c1;
```

- Method 2: Specify the table name and possibly skewed columns in SKEWJOIN HINT. In the following statement,

the c0 and c1 columns of Table a are skewed columns.

```
select /*+ skewjoin(a(c0, c1)) */ * from T0 a join T1 b on a.c0 = b.c0 and a.c1 = b.c1 and a.c2 = b.c2;
```

- Method 3: Specify the table name, columns, and skewed hot key values in SKEWJOIN HINT. If the skewed key values are of the STRING type, enclose each value with double quotation marks ("). In the following statement, (a.c0=1 and a.c1="2") and (a.c0=3 and a.c1="4") contain skewed hot key values.

```
select /*+ skewjoin(a(c0, c1)((1, "2"), (3, "4"))) */ * from T0 a join T1 b on a.c0 = b.c0 and a.c1 = b.c1 and a.c2 = b.c2;
```

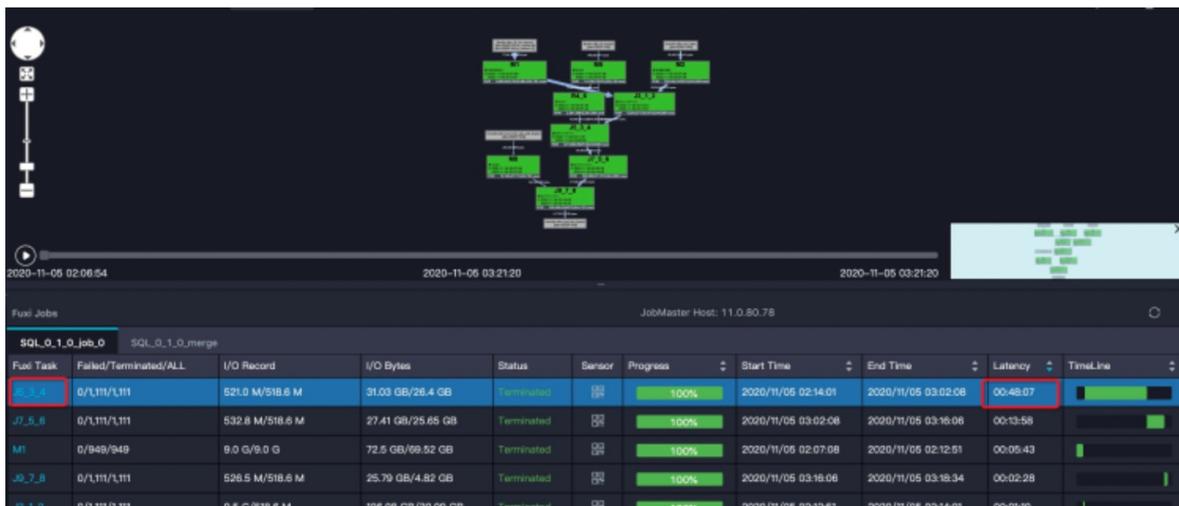
Differences

- Method 3 in which the skewvalue parameter is specified is more efficient than Method 1 and Method 2 in which the skewvalue parameter is not specified.
- If you use Analyze and Freeride to collect TopK skewed hot key values, SKEWJOIN may automatically take effect without requiring you to specify a hint.

Examples

1. Identify the JOIN statement that causes data skew.

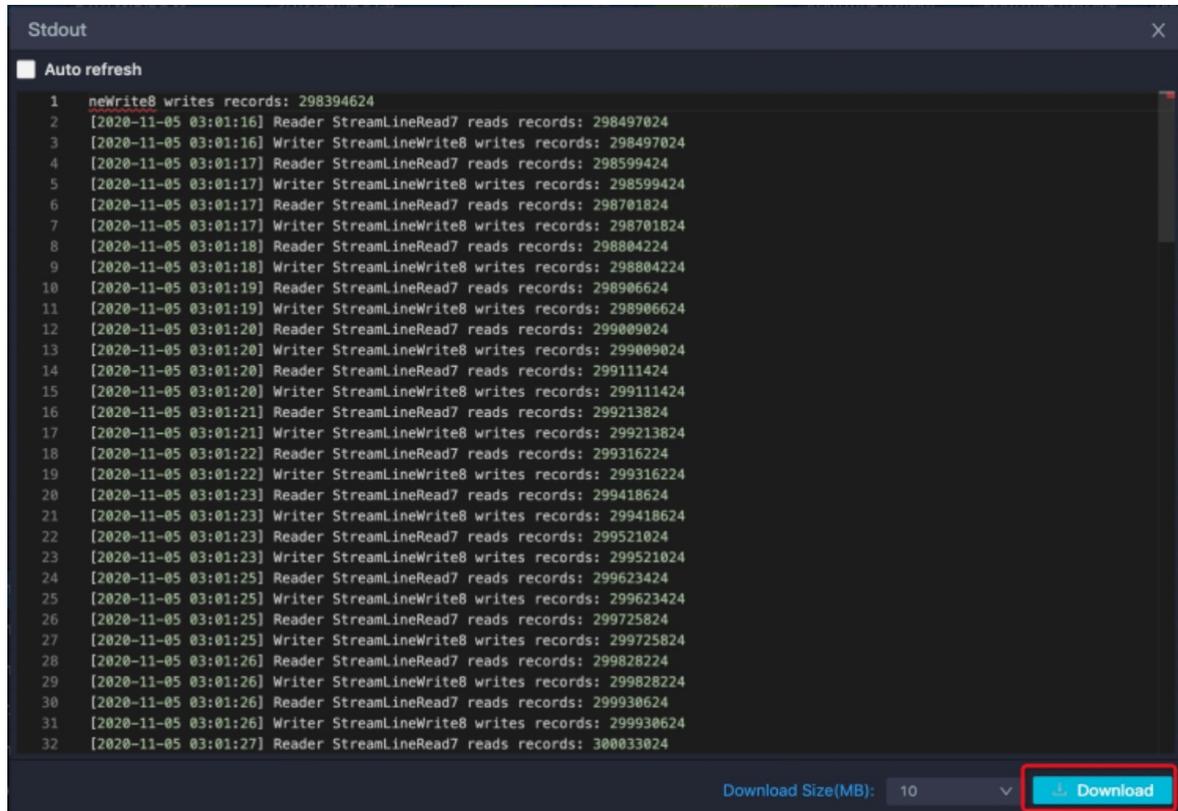
In the following screenshot captured on Logview, J5_3_4 is the Fuxi task that took the longest time to run.



Click the J5_3_4 task and query the instances of this task on the tab that appears. The query results show that the J5_3_4#215_0 instance takes the longest time to run and its I/O records and I/O bytes are much more than those of other instances.

No.	ID	IP	Path	Sensor	I/O Record	I/O Bytes	Status	StdOut	StdErr	Debug	Progress	Start Time	End Time	Latency	Rerun
240	J5_3_4#215_0	11.169.130.188			304.4 M/304.4 M	16.91 GB/17.56 GB	Terminated				100%	2020/11/05 02:14:03	2020/11/05 03:02:08	00:48:06	0
392	J5_3_4#352_0	11.169.130.34			68.3 M/69.3 M	3.7 GB/3.39 GB	Terminated				100%	2020/11/05 02:14:03	2020/11/05 02:19:36	00:05:33	0
60	J5_3_4#1053_0	11.169.128.163			3.5 M/3.5 M	189.12 MB/177.91 MB	Terminated				100%	2020/11/05 02:14:03	2020/11/05 02:14:25	00:00:22	0
328	J5_3_4#295_0	11.138.71.204			3.5 M/3.4 M	173.95 MB/164.04 ...	Terminated				100%	2020/11/05 02:14:04	2020/11/05 02:14:25	00:00:21	0
505	J5_3_4#454_0	11.169.140.143			77.4 K/75.3 K	6.35 MB/2.74 MB	Terminated				100%	2020/11/05 02:14:06	2020/11/05 02:14:23	00:00:17	0
285	J5_3_4#238_0	11.0.80.43			2.1 M/2.1 M	86.44 MB/92.05 MB	Terminated				100%	2020/11/05 02:14:04	2020/11/05 02:14:20	00:00:16	0
482	J5_3_4#442_0	11.169.93.219			87.4 K/85.3 K	7.1 MB/3.39 MB	Terminated				100%	2020/11/05 02:14:06	2020/11/05 02:14:22	00:00:16	0
583	J5_3_4#524_0	11.169.132.14			2.8 M/2.8 M	127.38 MB/117.91 MB	Terminated				100%	2020/11/05 02:14:03	2020/11/05 02:14:18	00:00:15	0

This indicates that data skew occurs on the J5_3_4#215_0 instance. However, the JOIN statement that causes the data skew needs to be further determined. You can open the stdout of the skewed instance and the stdout of a non-skewed instance. The StdOut of the skewed instance is enclosed in the yellow rectangle in the preceding figure. The Stdout dialog box shown in the following figure appears. In most cases, the web page cannot display all the content of stdout. To view all the content, you can click **Download** in this dialog box.



The following figures show that the value of record count in StreamLineRead7 of the skewed instance is much greater than that of the non-skewed instance. Therefore, data skew occurs when data in StreamLineWrite7 and StreamLineRead7 is shuffled.

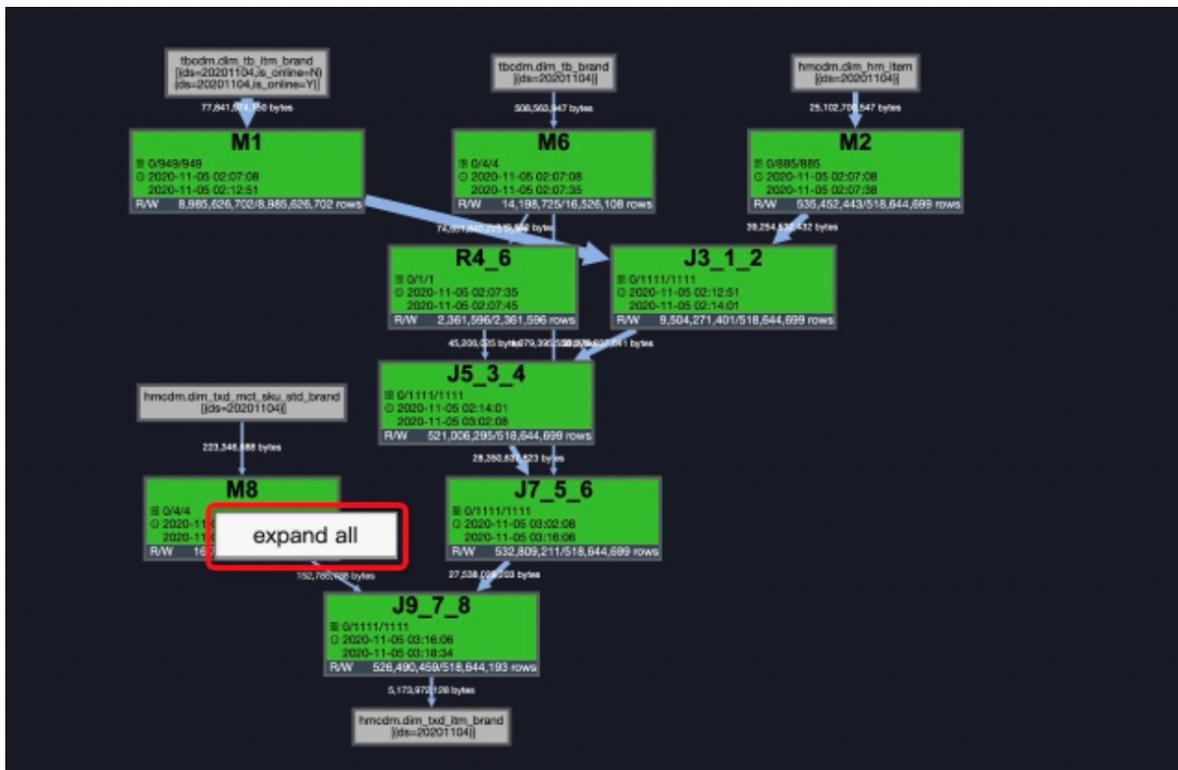
Skewed instance

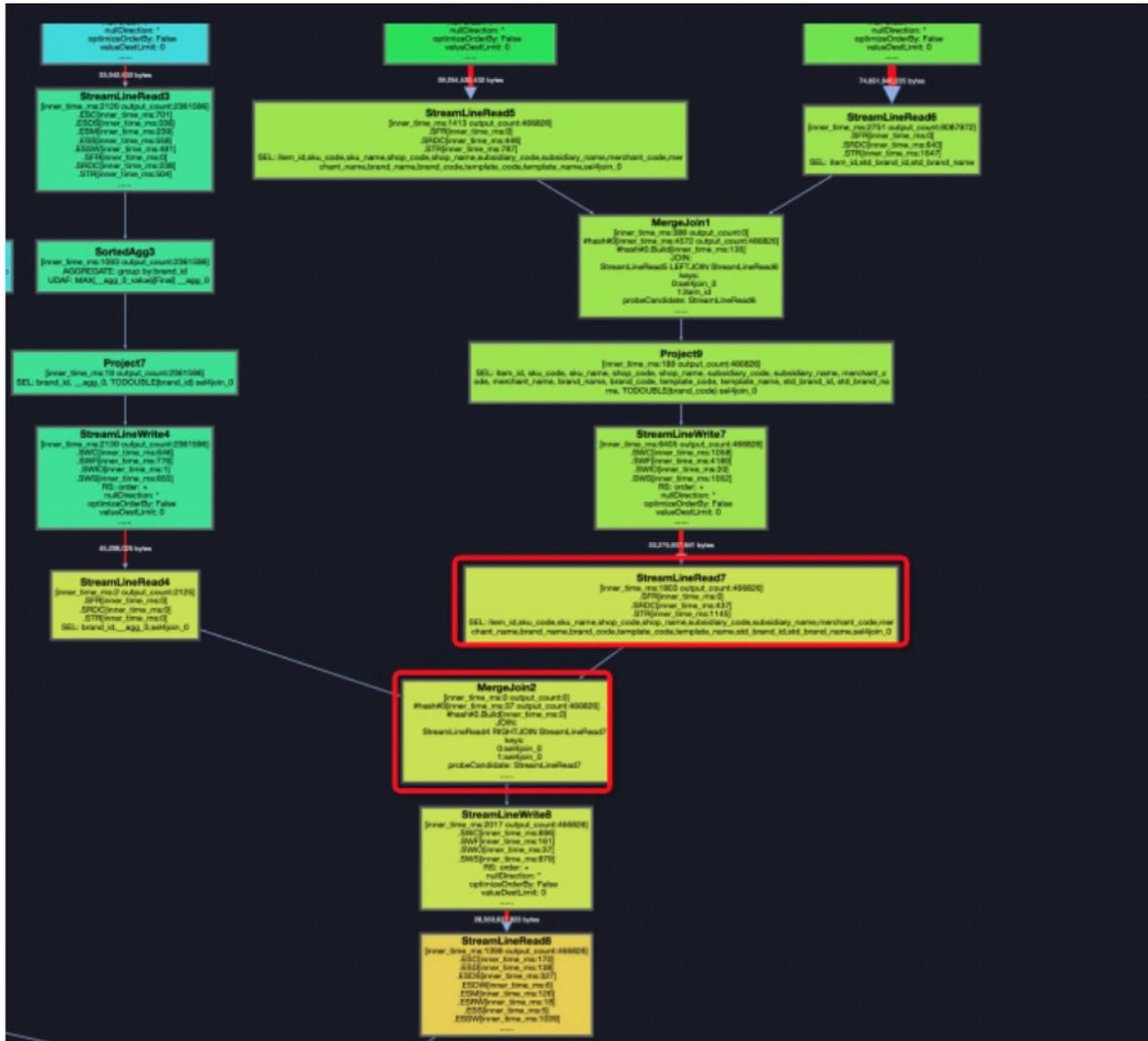
```
[2020-11-05 02:14:04] ----- Run Task: J5_3_4#215 -----
[2020-11-05 02:14:04] Reader StreamLineRead4, total estimated size: 69678, total record count: 2200
[2020-11-05 02:14:04] Reader StreamLineRead7, total estimated size: 55478096537, total record count: 204442049
[2020-11-05 02:14:06] Reader StreamLineRead4 reads records: 1024
[2020-11-05 02:14:06] Hash Join cursor MergeJoin2#Hash#9 loaded small table: {
  "EqualGroupCount": 2200,
  "ExitStartTimeStamp": 1604513646,
  "MaxBucketListDepth": 5,
  "MaxRowCountInSingleGroup": 1,
  "MemoryUsedComplex(MB)": 0,
  "MemoryUsedFixed(MB)": 0,
  "MemoryUsedHashTable(MB)": 0,
  "MemoryUsedString(MB)": 0,
  "SmallTableInputCount": 2200} duration: 0
[2020-11-05 02:14:06] Reader StreamLineRead7 reads records: 1024
[2020-11-05 02:14:06] Writer StreamLineWrite8 writes records: 1024
[2020-11-05 02:14:06] Reader StreamLineRead7 reads records: 102404
```

Non-skewed instance

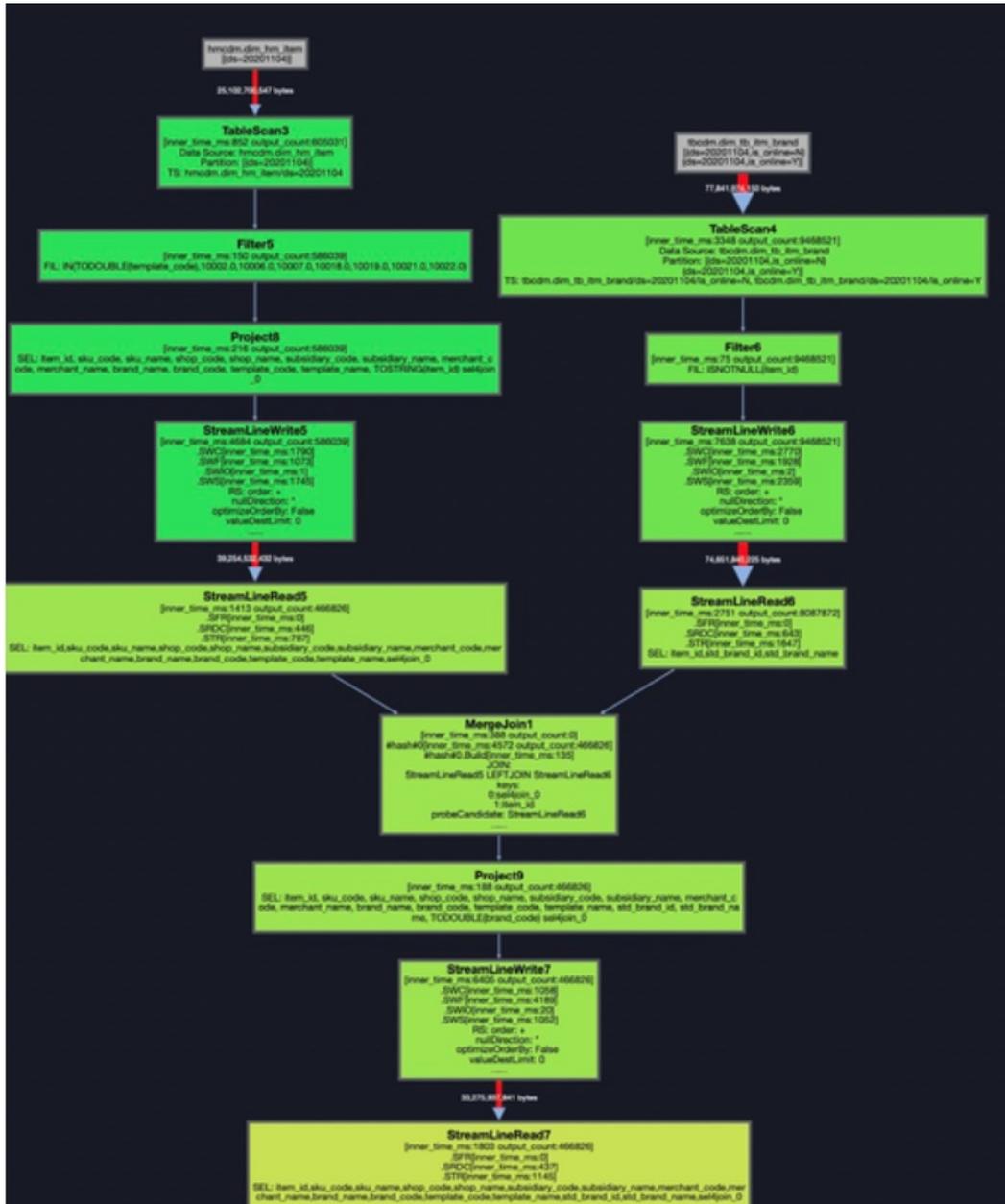
```
[2020-11-05 02:14:04] ----- Run Task: J5_3_4#1053 -----
[2020-11-05 02:14:04] Reader StreamLineRead4, total estimated size: 69195, total record count: 2150
[2020-11-05 02:14:04] Reader StreamLineRead7, total estimated size: 605691502, total record count: 3534181
[2020-11-05 02:14:06] Reader StreamLineRead4 reads records: 1024
[2020-11-05 02:14:06] Hash Join cursor MergeJoin2#hash#0 loaded small table: {
  "EqualGroupCount": 2150,
  "InitStartTimestamp": 1604513646,
  "MaxBucketListDepth": 5,
  "MaxRowCountInSingleGroup": 1,
  "MemoryUsedComplex(MB)": 0,
  "MemoryUsedFixed(MB)": 0,
  "MemoryUsedHashTable(MB)": 0,
  "MemoryUsedString(MB)": 0,
  "SmallTableInputCount": 2150} duration: 0
[2020-11-05 02:14:06] Reader StreamLineRead7 reads records: 1024
[2020-11-05 02:14:06] Writer StreamLineWrite0 writes records: 1024
[2020-11-05 02:14:06] Reader StreamLineRead7 reads records: 103424
[2020-11-05 02:14:06] Writer StreamLineWrite0 writes records: 103424
[2020-11-05 02:14:07] Reader StreamLineRead7 reads records: 205824
[2020-11-05 02:14:07] Writer StreamLineWrite0 writes records: 205824
[2020-11-05 02:14:07] Reader StreamLineRead7 reads records: 308224
[2020-11-05 02:14:07] Writer StreamLineWrite0 writes records: 308224
```

On the directed acyclic graph (DAG), you can right-click the skewed instance and select **expand all** to find StreamLineWrite7 and StreamLineRead7.





The DAG shows that data skew occurs on StreamLineRead7 in MergeJoin2. MergeJoin2 is generated after the dim_tm_item and dim_tm_brand tables are joined, and then the generated new table and the dim_tm_brand table are joined.



2. Locate the join based on the names of the related tables in Step 1. The result shows that data skew occurs in LEFT OUTER JOIN in the SQL statements and table t1 has data skew. To resolve this data skew issue, add the following command to the SQL statements:

```
/*+ skewjoin(t1) */
```

```

29     t1.sku_code,
30     t1.sku_name,]
31     t1.brand_code,
32     t1.brand_name,
33     t2.std_brand_id,
34     t2.std_brand_name
35 from hmcdm.dim_hm_item t1
36 left outer join tbcdm.dim_tb_itm_brand t2
37 on cast(t1.item_id as string)=cast(t2.item_id as string)
38 and t2.ds=max_pt('tbcdm.dim_tb_itm_brand')
39 where t1.ds='20201104'
40 and t1.template_code in (
41     10002,
42     10006,
43     10007,
44     10018,
45     10019,
46     10021,
47     10022
48 )
49 ) t1
50 left outer join(
51     select brand_id as std_brand_id, max(brand_name) as std_brand_name
52     from tbcdm.dim_tb_brand
53     where ds=max_pt('tbcdm.dim_tb_brand')
54     and is_standard_brand='Y'
55     and cast(brand_id as bigint)>0
56     group by brand_id
57 ) t21
58 on t1.brand_code=t21.std_brand_id
59 left outer join(
60     select

```

Usage notes

- Supported join types: INNER JOIN allows you to specify a hint for either the right or left table of the JOIN operation. LEFT JOIN, SEMI JOIN, and ANTI JOIN allow you to specify a hint only for the left table. RIGHT JOIN allows you to specify a hint only for the right table. FULL JOIN does not support SKEWJOIN HINT.
- An Aggregate is run after SKEWJOIN HINT is added, which slows down the JOIN operation. Therefore, we recommend that you add SKEWJOIN HINT only to the JOIN statements that cause data skew.
- The data type of Left Side Join Key must be the same as that of Right Side Join Key for the JOIN statement to which SKEWJOIN HINT is added. If the data types are different, SKEWJOIN HINT becomes ineffective.
- After a hint is added, the optimizer runs an Aggregate to dynamically obtain the hot key values of the first 20 rows with the most duplicate key values. 20 is a default value. You can add the following flag parameter and specify the number of the rows whose hot key values you want to return.

```
set odps.optimizer.skew.join.topk.num = xx;
```

- You can specify a hint only for one of the two tables in the JOIN operation.
- In the JOIN statement to which SKEWJOIN HINT is added, left key = right key must be included. SKEWJOIN HINT cannot be added to a CARTESIAN JOIN statement.
- The following statement shows how to use SKEWJOIN HINT with other hints. Note that SKEWJOIN HINT cannot be added to a JOIN statement to which MAPJOIN HINT is added.
- On the Json Summary tab of Logview, you can search for the topk_agg field to check whether SKEWJOIN HINT takes effect. If such a field exists, SKEWJOIN HINT takes effect.

6.17.2. Insufficient computing resources

This topic describes the issue of insufficient computing resources and provides a solution.

Computing resources in MaxCompute may be insufficient due to improper planning and misuse of cluster resources.

Jobs with insufficient computing resources have the following common characteristics:

- The job output stops at a stage and the progress remains unchanged. For example, in the following figure, the progress of M1_Stg1 in the job remains at 0%. R2_1_Stg1 depends on M1_Stg1. Therefore, the progress of R2_1_Stg1 remains at 0% before M1_Stg1 is complete.

```

2016-01-29 13:52:09 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:14 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:19 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:24 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:29 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:34 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:39 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:44 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:49 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:54 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:52:59 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:04 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:09 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:15 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:20 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:25 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:30 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:35 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:40 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:45 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:50 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
2016-01-29 13:53:55 M1_Stg1_job0:0/0/5 [0%] R2_1_Stg1_job0:0/0/1 [0%]
    
```

- Task instances are in the Ready state on Logview, as shown in the following figure. This state indicates that resources are insufficient to run these task instances. After an instance obtains the necessary resources, its state changes to Running. If an instance is in the Waiting state, the instance waits for the completion of the task on which it depends.

	FuxiInstanceID	IP & Path	StdOut	StdErr	Status
1	Odps/odps_s...				Ready
2	Odps/odps_s...				Ready
3	Odps/odps_s...				Ready
4	Odps/odps_s...				Ready
5	Odps/odps_s...				Ready

Each job is split into multiple tasks based on an execution plan. These tasks form a directed acyclic graph (DAG). Multiple instances are concurrently invoked in each task to complete the computation. In most cases, the resources required to invoke an instance are a single-core CPU and 2 GB memory. To properly allocate resources, a quota group is assigned to each project. The quota group determines the upper limit of resources (CPU and memory) that are available for all jobs in the project. If the resources used by the jobs that concurrently run reach the upper limit of the quota group, the jobs stop running due to insufficient resources.

You can use one of the following methods to resolve this issue:

- Run jobs in off-peak hours.
- Contact O&M personnel to increase the resource capacity of the quota group for the project.

6.17.3. Methods to optimize storage in MaxCompute SQL

This topic describes the methods to optimize storage in MaxCompute SQL.

Reasonably configure partitioned tables

MaxCompute supports the concept of partitioning in a table. A partition refers to a partition space that is specified when a table is created. That is, a few fields in the table serve as partition key columns. In most cases, you can consider a partition as a directory in a file system. Each value of a partition key column is called a partition in MaxCompute. You can group multiple fields of a table to a single partition to create a multi-level partition. Multi-level partitions are similar to multi-level directories. If you specify the name of a partition you want to access, the system reads data only from that partition and does not scan the entire table. This reduces costs and improves efficiency.

Example of a partitioned table:

```
create table src (key string, value bigint) partitioned by (pt string);
select * from src where pt='20160901';
-- Specifies the partitioning format. MaxCompute takes only the data in the 20160901 partition as the
input when MaxCompute generates a query plan.
```

Example of a non-partitioned table:

```
select * from src where key = 'MaxCompute';
-- The entire table is scanned in a query plan.
```

You can configure partitions by date or geographical region. You can also configure partitions based on your business requirements. Example:

```
create table if not exists sale_detail(
shop_name      string,
customer_id    string,
total_price    double)
partitioned by (sale_date string, region string);
-- Create a two-level partitioned table, in which level-1 partitions are specified by sale_date, and
level-2 partitions are specified by region.
```

Reasonably configure the table lifecycle

Storage resources in MaxCompute are valuable. You can configure the lifecycle of a table based on data usage. MaxCompute deletes data that exceeds the lifecycle threshold at the earliest opportunity to save storage space.

Example:

```
create table test3 (key boolean) partitioned by (pt string, ds string) lifecycle 100;
-- Create a table with a lifecycle of 100 days. If the last modification of the table or a partition
occurred more than 100 days ago, MaxCompute deletes the table or partition.
```

 **Notice** The lifecycle takes a partition as the smallest unit. For a partitioned table, if some partitions reach the lifecycle threshold, they will be directly deleted. Partitions that have not reached the lifecycle threshold are not affected.

You can run the following command to change the lifecycle of a created table:

```
alter table table_name set lifecycle days;
```

Archive cold data

Some data needs to be retained permanently or for a long period of time. However, the frequency of accessing the data decreases over time. If the use frequency decreases to an extent, you can archive the data as RAID files. Data is not stored as three copies. The Cauchy Reed-Solomon algorithm is used to store data as six copies of the original data plus three parity blocks. This improves the effective storage ratio from 1:3 to 1:1.5. MaxCompute uses the bzip2 algorithm to archive tables with a higher data compression ratio than other algorithms. These two algorithms can be combined to save the storage space by more than 70%.

Format of an archiving command:

```
ALTER TABLE table_name [PARTITION(partition_name='partition_value')] ARCHIVE;
```

Example:

```
alter table my_log partition(ds='20170101') archive;
-- Archive the data with ds set to 20170101.
```

Merge small files

A large number of small files are generated during the Reduce computing process or during Tunnel-based real-time data collection. This causes the following issues:

- More instance resources are required because the number of files a single instance can process is limited. This results in a waste of resources and affects the overall execution performance.
- The workload on the file system is high, and the disk utilization is affected.

You can use one of the following methods to merge small files:

- **ALTER command:** Run the following command in the console command line to merge small files:

```
ALTER TABLE tablename [PARTITION] MERGE SMALLFILES;
```

- **SQL statement:** After SQL statements are executed, set `odps.task.merge.enabled` to true and run a Fuxi job to merge small files.

6.17.4. UDF OOM

This topic describes the UDF OOM issue and provides a solution.

An OOM error is reported when some jobs are running. Detailed error information:

```
FAILED: ODPS-0123144: Fuxi job failed - WorkerRestart errCode:9,errMsg:SigKill(OOM), usually caused
by OOM(out of memory)
```

You can configure UDF runtime parameters to fix the error. Example:

```
odps.sql.mapper.memory=3072;
set odps.sql.udf.jvm.memory=2048;
set odps.sql.udf.python.memory=1536;
```

6.18. Limits of MaxCompute SQL statements

This topic describes all the limits of MaxCompute SQL statements.

Limit	Maximum value/Limit	Requirement	Description
Table name length	128 bytes	Size	A table or column name can contain only letters, digits, and underscores (_). It must start with a letter.
Comment length	1,024 bytes	Size	A comment is a valid string that cannot exceed 1,024 bytes in length.
Column definitions in a table	1,200	Quantity	A table can contain a maximum of 1,200 column definitions.
Partitions in a table	60,000	Quantity	A table can contain a maximum of 60,000 partitions.
Partition levels of a table	6	Quantity	A table can contain a maximum of six levels of partitions.
Destination INSERT objects	256	Quantity	In a MULTI-INSERT operation, you can insert data into a maximum of 256 tables at a time.
UNION ALL	256	Quantity	A UNION ALL operation can be performed on a maximum of 256 tables.
MAPJOIN	128	Quantity	A MAPJOIN operation can be performed on a maximum of 128 small tables.
MAPJOIN memory	512 MB	Size	The memory size for all small tables on which a MAPJOIN operation is performed cannot exceed 512 MB.
Window functions	5	Quantity	A SELECT statement can contain a maximum of five window functions.
PT IN SUBQUERY	1,000	Quantity	A PT IN SUBQUERY statement can generate a maximum of 1,000 rows.
Length of an SQL statement	2 MB	Size	An SQL statement cannot exceed 2 MB in length. This limit is suitable for the scenarios where you use an SDK to call SQL statements.

Limit	Maximum value/Limit	Requirement	Description
Conditions of a <code>WHERE</code> clause	256	Quantity	A <code>WHERE</code> clause can contain a maximum of 256 conditions.
Length of a column record	8 MB	Size	A column record in a table cannot exceed 8 MB in length.
Parameters in an <code>IN</code> clause	1,024	Quantity	This item specifies the maximum number of parameters in an <code>IN</code> clause, for example, <code>IN (1,2,3...,1024)</code> . If the number of parameters in an <code>IN</code> clause is too large, the compilation performance is affected. We recommend that you use no more than 1,024 parameters, but this is not a fixed upper limit.
<code>jobconf.json</code>	1 MB	Size	The maximum size of the <code>jobconf.json</code> file is 1 MB. If a table contains a large number of partitions, the size of the <code>jobconf.json</code> file may exceed 1 MB.
View	Not writable	Operation	A view is not writable and does not support the <code>INSERT</code> operation.
Data type and position of a column	Unmodifiable	Operation	The data type and position of a column cannot be modified.
Java UDFs	Not allowed to be <code>abstract</code> or <code>static</code>	Operation	Java UDFs cannot be <code>abstract</code> or <code>static</code> .
Partitions that can be queried	10,000	Quantity	A maximum of 10,000 partitions can be queried.
SQL execution plans	1 MB	Size	The size of the execution plan generated by MaxCompute SQL cannot exceed 1 MB. Otherwise, <code>FAILED: ODPS-0010000:System internal error - The Size of Plan is too large</code> is returned.

 **Notice** The preceding limits cannot be manually modified.

6.19. Appendix

6.19.1. Escape characters

This topic describes the escape characters in MaxCompute SQL.

In MaxCompute SQL, the backslash (\) is an escape character that invokes an alternative representation on the special characters in a character string. This escape character may also indicate a literal interpretation of the characters that follow the escape character. If the backslash (\) in a string constant is followed by three valid octal digits in the range from 001 to 177, the system converts the ASCII values to the corresponding characters. The following table describes the mapping between escape characters and represented special characters.

Escape character	Represented special character
\b	Backspace
\t	Tab
\n	Newline
\r	Carriage-return
\'	Single quotation mark
\"	Double quotation marks
\\	Backslash
\;	Semicolon
\Z	Control-Z
\0 or \00	Ending character

Examples:

- Use an escape character to indicate a literal interpretation of the characters that follow the escape character.

String constants in MaxCompute SQL can be expressed in single quotation marks (') or double quotation marks ("). You can include double quotation marks (") in a string that is enclosed in single quotation marks (') or include single quotation marks (') in a string that is enclosed in double quotation marks ("). Examples:

```
"I'm a happy manong."
'I\'m a happy manong.'
```

- Use an escape character to represent special characters. Examples:

```
select length('a\tb');
-- 3 is displayed in the returned result. This indicates that the string contains three characters
and that \t is treated as one character.
```

```
select 'a\ab',length('a\ab');
-- aab, 3 is displayed in the returned result. \a is treated as letter a.
```

6.19.2. LIKE usage

This topic describes how to use the LIKE operator for character matching.

In LIKE character matching, a percent sign (%) is a wildcard character that matches an arbitrary number of characters. An underscore (_) is a wildcard character that matches a single character.

To match percent signs (%) or underscores (_), escape them with double backslashes (\\). Specifically, \\% or _ is used for the match.

```
'abcd' like 'ab%'
-- True.
'abcd' like 'ab_'
-- False.
'ab_cde' like 'ab\\_c%';
-- True.
```

 **Note** MaxCompute SQL statements support only UTF-8 character sets. If data is encoded in another format, the calculation result may be incorrect.

6.19.3. Regular expressions

This topic describes regular expressions in MaxCompute SQL.

Regular expressions in MaxCompute SQL use the Perl Compatible Regular Expressions (PCRE) standards and are matched by character.

RLIKE

You can use the RLIKE statement to match regular expressions. The following table lists the supported metacharacters.

Metacharacter	Description
^	Match the beginning of a line.
\$	Match the end of a line.
.	Match any characters.
*	Match zero or more instances of the preceding character or character pattern.
+	Match one or more instances of the preceding character or character pattern.
?	Match zero or one instance of the preceding character or character pattern.
?	Match a modifier. If this character follows one of other delimiters (*, +, ?, {n}, {n,}, or {n,m}), the match pattern is non-greedy. In the non-greedy pattern, as few characters as possible are matched with the searched character string. In the default greedy pattern, as many characters as possible are matched with the searched character string.
A B	Match A or B.
(abc)*	Match zero or more instances of the abc sequence.
{n} or {m,n}	The number of matches.
[ab]	Match any character (a or b) in the brackets. Fuzzy match is supported.
[a-d]	Match any of the following characters: a, b, c, and d.
[^ab]	Match any character that is not a or b. ^ indicates 'non'.
:::	For more information, see the following table.
\	The escape character.

Metacharacter	Description
\n	n indicates a digit from 1 to 9 and is backward referenced.
\d	Digits.
\D	Non-digit characters.

POSIX character groups

Character group	Description	Valid value
[:alnum:]	Letters and digits	[a-zA-Z0-9]
[:alpha:]	Letters	[a-zA-Z]
[:ascii:]	ASCII characters	[\x00-\x7F]
[:blank:]	Spaces and tab characters	[\t]
[:cntrl:]	Control characters	[\x00-\x1F\x7F]
[:digit:]	Digits	[0-9]
[:graph:]	Characters other than whitespace characters	[\x21-\x7E]
[:lower:]	Lowercase letters	[a-z]
[:print:]	[:graph:] and whitespace characters	[\x20-\x7E]
[:punct:]	Punctuation marks	[!\"#\$%&'()*+,-./:;<=>?@^_`{ }~]
[:space:]	Whitespace characters	[\t\r\n\v\f]
[:upper:]	Uppercase letters	[A-Z]
[:xdigit:]	Hexadecimal characters	[A-Fa-f0-9]

Escape characters

The system uses a backslash (\) as the escape character. A backslash (\) in a regular expression indicates a second escape. For example, a regular expression is used to match the `a+b` string. The plus sign (`+`) is a special character in the expression and must be escaped. In the regular expression engine, the string is expressed as `a\\+b`. The system must perform an escape on the expression. Therefore, the expression that can match the string is `a\\\\+b`.

Assume that the `test_dual` table exists. Example:

```
select 'a+b' rlike 'a\\\\+b' from test_dual;
```

Returned result:

```
+-----+
| _c1 |
+-----+
| true |
+-----+
```

The \ character is a special character in the regular expression engine. To match this character, it is expressed as \\ in the engine. Then, the system performs an escape on the expression. As a result, the \ character is expressed as \\.

```
select 'a\\b', 'a\\b' rlike 'a\\\\b' from test_dual;
```

Returned result:

```
+-----+-----+
| _c0 | _c1 |
+-----+-----+
| a\b | false |
+-----+-----+
```

 **Note** If a MaxCompute SQL statement includes `a\\b`, `a\b` is displayed in the output because MaxCompute escapes the expression.

If a string includes tab characters, the system reads `\t` and stores it as one character. Therefore, it is a common character in regular expressions.

```
select 'a\tb', 'a\tb' rlike 'a\tb' from test_dual;
```

Returned result:

```
+-----+-----+
| _c0      | _c1 |
+-----+-----+
| a      b | true |
+-----+-----+
```

6.19.4. Reserved words and keywords

This topic describes all reserved words and keywords in MaxCompute SQL.

Notice

- When you name a table, a column, or a part ition, do not use reserved words or keywords. Otherwise, an error may occur.
- Reserved words are not case-sensitive.

```

%      &      &&      (      )      *      +
-      .      /      ;      <      <=      <>
=      >      >=      ?      ADD      AFTER      ALL
ALTER      ANALYZE      AND      ARCHIVE      ARRAY      AS      ASC
BEFORE      BETWEEN      BIGINT      BINARY      BLOB      BOOLEAN      BOTH      DECIMAL
BUCKET      BUCKETS      BY      CASCADE      CASE      CAST      CFILE
CHANGE      CLUSTER      CLUSTERED      CLUSTERSTATUS      COLLECTION      COLUMN      COLUMNS
COMMENT      COMPUTE      CONCATENATE      CONTINUE      CREATE      CROSS      CURRENT
CURSOR      DATA      DATABASE      DATABASES      DATE      DATETIME      DBPROPERTIES
DEFERRED      DELETE      DELIMITED      DESC      DESCRIBE      DIRECTORY      DISABLE
DISTINCT      DISTRIBUTE      DOUBLE      DROP      ELSE      ENABLE      END      EXCEPT
ESCAPED      EXCLUSIVE      EXISTS      EXPLAIN      EXPORT      EXTENDED      EXTERNAL
FALSE      FETCH      FIELDS      FILEFORMAT      FIRST      FLOAT      FOLLOWING
FORMAT      FORMATTED      FROM      FULL      FUNCTION      FUNCTIONS      GRANT
GROUP      HAVING      HOLD_DDLTIME      IDXPROPERTIES      IF      IMPORT      IN
INDEX      INDEXES      INPATH      INPUTDRIVER      INPUTFORMAT      INSERT      INT
INTERSECT      INTO      IS      ITEMS      JOIN      KEYS      LATERAL
LEFT      LIFECYCLE      LIKE      LIMIT      LINES      LOAD      LOCAL
LOCATION      LOCK      LOCKS      LONG      MAP      MAPJOIN      MATERIALIZED
MINUS      MSCK      NOT      NO_DROP      NULL      OF      OFFLINE      OFFSET
ON      OPTION      OR      ORDER      OUT      OUTER      OUTPUTDRIVER
OUTPUTFORMAT      OVER      OVERWRITE      PARTITION      PARTITIONED      PARTITIONPROPERTIES      PARTITIO
NS
PERCENT      PLUS      PRECEDING      PRESERVE      PROCEDURE      PURGE      RANGE
RCFILE      READ      READONLY      READS      REBUILD      RECORDREADER      RECORDWRITER
REDUCE      REGEXP      RENAME      REPAIR      REPLACE      RESTRICT      REVOKE
RIGHT      RLIKE      ROW      ROWS      SCHEMA      SCHEMAS      SELECT
SEMI      SEQUENCEFILE      SERDE      SERDEPROPERTIES      SET      SHARED      SHOW
SHOW_DATABASE      SMALLINT      SORT      SORTED      SSL      STATISTICS      STATUS      STORED
STREAMTABLE      STRING      STRUCT      TABLE      TABLES      TABLESAMPLE      TBLPROPERTIES
TEMPORARY      TERMINATED      TEXTFILE      THEN      TIMESTAMP      TINYINT      TO
TOUCH      TRANSFORM      TRIGGER      TRUE      TYPE      UNARCHIVE      UNBOUNDED      UNDO
UNION      UNIONTYPE      UNIQUEJOIN      UNLOCK      UNSIGNED      UPDATE      USE
USING      UTC      UTC_TMESTAMP      VIEW      WHEN      WHERE      WHILE      DIV
    
```

6.19.5. Settings of new data types

This topic describes the precautions for configuring new data types in MaxCompute SQL.

If you want to read a table that includes new data types, you are not required to add the

```
setodps.sql.type.system.odps2=true;
```

flag. However, you must take note of the following points:

- If the flag is not added, the read data is implicitly converted into the original data type for all computations.
- If the flag is not added for integer constants, the BIGINT type is used, and an error message is reported.
- If you write data to a table and the data is in passthrough mode, you can choose not to add the new data type flag. However, if you want to calculate the data, an error is reported because the implicit data type conversion is invalid.

6.19.6. ACID semantics of MaxCompute parallel write jobs

This topic describes the atomicity, consistency, isolation, durability (ACID) semantics of MaxCompute parallel write jobs.

Terms

- Operation: a single job submitted in MaxCompute.
- Data object: an object that stores data, such as a non-partitioned table or a partition.
- INTO job: an SQL job that contains the INTO keyword, such as INSERT INTO or DYNAMIC INSERT INTO.
- OVERWRITE job: an SQL job that contains the OVERWRITE keyword, such as INSERT OVERWRITE or DYNAMIC INSERT OVERWRITE.
- Data upload by using Tunnel: an INTO or OVERWRITE job.

Description of ACID semantics

- Atomicity: An operation is completely performed or not performed at all. An operation is not partially performed.
- Consistency: The integrity of data objects is maintained when an operation is performed.
- Isolation: An operation is independent of other parallel operations.
- Durability: After an operation is complete, modified data is permanently valid and not lost even if a system failure occurs.

Scenarios of ACID semantics for MaxCompute

- Atomicity
 - When multiple jobs conflict with each other, MaxCompute ensures that only one job succeeds.
 - The atomicity of the CREATE, OVERWRITE, and DROP operations on a single table or partition is ensured.
 - The atomicity of cross-table operations, such as MULTI-INSERT, cannot be ensured.
 - In extreme cases, the following operations may not be atomic:
 - A `DYNAMIC INSERT OVERWRITE` operation that is performed on more than 10,000 partitions.
 - An INTO operation. The atomicity of INTO operations cannot be ensured because data cleansing fails during a transaction rollback. However, the data cleansing failure does not cause loss of original data.
- Consistency
 - The consistency can be ensured for OVERWRITE jobs.
 - If an INTO job fails due to a conflict, data from the failed job may remain.
- Isolation
 - For non-INTO operations, MaxCompute ensures that read operations are submitted.
 - For INTO operations, some read operations may not be submitted.
- Durability
 - MaxCompute ensures data durability.

ACID semantics of transactional tables

- If existing ACID semantics are used, INTO operations ensure that read operations are submitted and no data remains after an operation fails due to a conflict.
- The atomicity of the UPDATE and DELETE operations on a single non-partitioned table or a partition is ensured. For example, if two update jobs run in parallel to modify the data of a partition, only one job succeeds. The partition is not partially updated by one job. It is also not updated by the two jobs with part of the updated data remained.

Conflicts among parallel operations: When two jobs run in parallel and write data to the same destination table, a conflict may occur. In this case, the job that ends earlier will succeed, and the job that ends later will fail due to the conflict.

The following table describes the conflicts that may occur when two jobs run in parallel and write data to the same non-partitioned table or partition. The first column lists the job that ends earlier. The other columns list the jobs that end later.

Before/After	INSERT OVERWRITE	INSERT INTO	UPDATE/DELETE	MERGE SMALLFILES
INSERT OVERWRITE	Both jobs succeed. The result data of the INSERT OVERWRITE operation that ends later will overwrite that of the INSERT OVERWRITE operation that ends earlier.	Both jobs succeed. The result data of the INSERT INTO operation is appended to that of the INSERT OVERWRITE operation.	The INSERT OVERWRITE operation modifies the data of the non-partitioned table or partition. An error is reported in the UPDATE operation.	The INSERT OVERWRITE operation modifies the data of the non-partitioned table or partition. An error is reported in the MERGE SMALLFILES operation.
INSERT INTO	Both jobs succeed. The result data of the INSERT OVERWRITE operation overwrites that of the INSERT INTO operation.	Both jobs succeed. The result data of the two INSERT INTO operations is merged.	The INSERT INTO operation modifies the data of the non-partitioned table or partition. An error is reported in the UPDATE operation that ends later.	The INSERT INTO operation modifies the data of the non-partitioned table or partition. An error is reported in the MERGE SMALLFILES operation that ends later.
UPDATE/DELETE	Both jobs succeed. The result data of the INSERT OVERWRITE operation overwrites that of the UPDATE operation.	Both jobs succeed. The result data of the INSERT INTO operation is appended to that of the UPDATE operation.	The UPDATE operation that ends earlier modifies the data of the non-partitioned table or partition. An error is reported in the UPDATE operation that ends later.	The UPDATE operation modifies the data of the non-partitioned table or partition. An error is reported in the MERGE SMALLFILES operation that ends later.
MERGE SMALLFILES	Both jobs succeed. The result data of the INSERT OVERWRITE operation overwrites that of the MERGE SMALLFILES operation.	Both jobs succeed. The result data of the INSERT INTO operation is appended to that of the MERGE SMALLFILES operation.	The MERGE SMALLFILES operation modifies the data of the non-partitioned table or partition. An error is reported in the UPDATE operation that ends later.	The MERGE SMALLFILES operation that ends earlier modifies the data of the non-partitioned table or partition. An error is reported in the MERGE SMALLFILES operation that ends later.

The following table simplifies the preceding rules and describes only whether an error is reported for an operation after data changes.

Operation	Error reported upon data changes
INSERT OVERWRITE	No
INSERT INTO	No
UPDATE/DELETE	Yes
MERGE SMALLFILES	Yes

No conflict occurs for INSERT operations in the event of data changes. An error is reported in the UPDATE/DELETE or MERGE SMALLFILES operation if the destination non-partitioned table or partition has data changes.

 Note

- Minor or major compaction is classified as MERGE SMALLFILES operations.
- In extreme scenarios, if multiple jobs are all in the metadata update phase, an error may be reported due to a conflict among these parallel update operations.

7. MaxCompute Tunnel

7.1. Overview

This topic describes how to upload data to MaxCompute or download data from MaxCompute. This topic also describes related limits.

MaxCompute provides two types of channels for data uploads and downloads:

- **DataHub:** This channel is used to upload or download data in real time. It includes the OGG, Flume, Logstash, and Fluentd plug-ins.
- **Tunnel:** This channel is used to upload or download large amounts of data at a time. It includes the MaxCompute client, DataWorks, DTS, Sqoop, Kettle plug-in, and MaxCompute Migration Assist (MMA).

DataHub and Tunnel provide their own SDKs. A variety of data upload and download tools are derived from the SDKs.

The tools can meet the data upload and download requirements of most common scenarios in which data is migrated to the cloud.

Tunnel service connections

DataHub and Tunnel use different endpoints in different network environments. You must also select different endpoints to connect to the service.

Limits

- **Limits on Tunnel-based data uploads:**
 - You cannot run Tunnel commands to upload or download data of the ARRAY, MAP, or STRUCT type.
 - No limits are specified for the upload speed. The upload speed depends on the network bandwidth and server performance.
 - The number of retries is limited. If the number of retries exceeds the limit, the next block is uploaded. After data is uploaded, you can execute the `select count(*) from table_name` statement to check whether any data is lost.
 - By default, a project supports a maximum of 2,000 concurrent Tunnel connections.
 - On the server, the lifecycle of a session is 24 hours. A session can be shared among processes and threads on the server, but you must make sure that each block ID is unique.
 - MaxCompute ensures the validity of concurrent writes based on atomicity, consistency, isolation, durability (ACID).

- **Limits on DataHub-based data uploads:**
 - The size of each field cannot exceed its upper limit.

 **Note** The size of a string cannot exceed 8 MB.

- During an upload, multiple data records are packaged.
- **Limits on TableTunnel SDK interfaces:**
 - A block ID must be greater than or equal to 0 but less than 20,000. The size of the data that you want to upload in a block cannot exceed 100 GB.
 - The lifecycle of a session is 24 hours. If you want to transfer large amounts of data, more than 24 hours are required. In this case, we recommend that you transfer the data in multiple sessions.
 - The lifecycle of an HTTP request that corresponds to a RecordWriter is 120 seconds. If no data flows over an HTTP connection within 120 seconds, the server closes the connection.

7.2. Selection of tools to migrate data to the cloud

MaxCompute provides a variety of data upload and download tools, which can be used in different scenarios to migrate data to the cloud. This topic describes the selection of data transmission tools in three typical scenarios.

Hadoop data migration

You can use MaxCompute Migration Assist (MMA), Sqoop, and DataWorks to migrate Hadoop data.

- If you use DataWorks, DataX is required.
- If you use Sqoop, a MapReduce job runs on the original Hadoop cluster for distributed data transmission to MaxCompute.
- If you use MMA, Meta Carrier is required to access your Hive metastore service and capture Hive metadata. Then, MMA generates DDL statements that are used to create MaxCompute tables and partitions, as well as Hive UDTF SQL statements for data migration.

Synchronization of data in a database

To synchronize data from a database to MaxCompute, you must select a tool based on the database type and synchronization policy.

- Use DataWorks for offline batch synchronization. DataWorks supports a wide range of database types, such as MySQL, SQL Server, and PostgreSQL.
- Use the OGG plug-in for real-time synchronization of data in an Oracle database.
- Use DTS for real-time synchronization of data in an ApsaraDB RDS database.

Log collection

You can use tools such as Flume, Fluentd, and Logstash to collect logs.

7.3. Introduction to the tools

MaxCompute supports a wide range of data upload and download tools. The source code for most of the tools can be found and maintained on the open source community GitHub. You can select the appropriate tools to upload and download data based on usage scenarios. This topic describes these tools.

Alibaba Cloud services

- Data Integration of DataWorks (Tunnel)

Data Integration of DataWorks is a stable, efficient, and scalable data synchronization platform. It is designed to provide full offline and incremental real-time data synchronization, integration, and exchange services for the heterogeneous data storage systems on Alibaba Cloud.

Data synchronization tasks support the following data sources: MaxCompute, ApsaraDB RDS (MySQL, SQL Server, and PostgreSQL), Oracle, FTP, AnalyticDB, OSS, ApsaraDB for Memcache, and PolarDB-X.

- MaxCompute client (Tunnel)

Based on the batch data tunnel SDK, the client provides built-in Tunnel commands for data uploads and downloads.

- DTS (Tunnel)

Data Transmission Service (DTS) is an Alibaba Cloud data service that supports data exchange among multiple data sources, such as Relational Database Management System (RDBMS), NoSQL, and Online Analytical Processing (OLAP). It provides data transmission features, such as data migration, real-time data subscription, and real-time data synchronization.

DTS supports data synchronization from ApsaraDB RDS and MySQL instances to MaxCompute tables. Other data sources are not supported.

Open source products

The projects that correspond to each product are open-sourced. You can visit [aliyun-maxcompute-data-collectors](#) to view details.

- Sqoop (Tunnel)

Sqoop 1.4.6 in the community is further developed to provide enhanced MaxCompute support. It imports data from relational databases such as MySQL and data from HDFS or Hive to MaxCompute tables. It also exports data from MaxCompute tables to relational databases such as MySQL.

- Kettle (Tunnel)

Kettle is an open source extract, transform, load (ETL) tool that is developed in Java. It runs on Windows, UNIX, or Linux, and provides graphic interfaces for you to define the data transmission topology by using drag-and-drop components.

- Apache Flume (DataHub)

- Apache Flume is a distributed and reliable system. It efficiently collects large volumes of log data from different data sources and then aggregates and stores the data in a centralized data storage. It supports multiple Source and Sink plug-ins.
- The DataHub Sink plug-in of Apache Flume allows you to upload log data to DataHub in real time and archive the data in MaxCompute tables.

- Fluentd (DataHub)

- Fluentd is an open source software product. It collects logs, such as application logs, system logs, and access logs, from various sources. It allows you to use plug-ins to filter log data and store the data in different data processors, including MySQL, Oracle, MongoDB, Hadoop, and Treasure Data.
- The DataHub plug-in of Fluentd allows you to upload log data to DataHub in real time and archive the data in MaxCompute tables.

- Logstash (DataHub)

- Logstash is an open source log collection and processing framework. The logstash-output-datahub plug-in allows you to import data to DataHub. This tool can be easily configured to collect and transmit data. It can be used together with MaxCompute or StreamCompute to easily create an all-in-one streaming data solution from data collection to analytics.
- The DataHub plug-in of Logstash allows you to upload log data to DataHub in real time and archive the data in MaxCompute tables.

- OGG (DataHub)

The DataHub plug-in of OGG allows you to incrementally synchronize data in an Oracle database to DataHub in real time and archive the data in MaxCompute tables.

- MMA

MaxCompute Migration Assist (MMA) uses Meta Carrier to access your Hive metastore service and capture Hive metadata. It then generates DDL statements that are used to create MaxCompute tables and partitions, as well as Hive UDTF SQL statements for data migration.

7.4. Tunnel SDK overview

7.4.1. Overview

Data upload and download tools provided by MaxCompute are developed based on Tunnel SDK. This topic describes the major APIs of Tunnel SDK.

The usage of the SDK varies based on its version. For more information, see [SDK Java Doc](#).

API	Description
TableTunnel	The entry class that is used to access MaxCompute Tunnel.
TableTunnel.UploadSession	A session that uploads data to a MaxCompute table.
TableTunnel.DownloadSession	A session that downloads data from a MaxCompute table.
InstanceTunnel	The entry class that is used to access MaxCompute Tunnel.
InstanceTunnel.DownloadSession	A session that downloads data from a MaxCompute SQL instance. The SQL instance must start with the SELECT keyword and is used to query data.

Note

- If you use Maven, you can search for odps-sdk-core in the Maven repository to find the latest version of the SDK for Java. You can configure the Maven dependency in the following way:

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-core</artifactId>
  <version>0.36.2</version>
</dependency>
```

- The endpoint of MaxCompute Tunnel supports automatic routing based on the MaxCompute endpoint settings.

7.4.2. TableTunnel

This topic describes the TableTunnel API.

TableTunnel is an entry class of the MaxCompute Tunnel service. You can use TableTunnel to upload or download only table data. Views cannot be uploaded or downloaded.

Definition

The following code defines the TableTunnel API.

```
public class TableTunnel {
  public DownloadSession createDownloadSession(String projectName, String tableName);
  public DownloadSession createDownloadSession(String projectName, String tableName, PartitionSpec partitionSpec);
  public UploadSession createUploadSession(String projectName, String tableName);
  public UploadSession createUploadSession(String projectName, String tableName, PartitionSpec partitionSpec);
  public DownloadSession getDownloadSession(String projectName, String tableName, PartitionSpec partitionSpec, String id);
  public DownloadSession getDownloadSession(String projectName, String tableName, String id);
  public UploadSession getUploadSession(String projectName, String tableName, PartitionSpec partitionSpec, String id);
  public UploadSession getUploadSession(String projectName, String tableName, String id);
  public void setEndpoint(String endpoint);
}
```

Description

- The lifecycle of a Tunnel instance starts from the time it is created to the time data upload or download is complete.
- Tunnel provides a method to create UploadSession and DownloadSession objects. Tunnel.UploadSession is used to upload data. Tunnel.DownloadSession is used to download data.
- A session refers to the process of uploading or downloading a table or partition. A session consists of one or more HTTP requests to Tunnel RESTful APIs.
- In an upload session, each RecordWriter matches an HTTP request and is identified by a unique block ID. The block ID is the name of the file that corresponds to the RecordWriter.
- If you use the same block ID to enable a RecordWriter multiple times in the same session, the data uploaded after the RecordWriter calls the close() method for the last time overwrites all the data that is previously uploaded. This feature can be used to retransmit a data block that fails to be uploaded.
- In UploadSession of Tunnel:
 - If the boolean overwrite parameter is not specified, the INSERT INTO statement is used.
 - If the boolean overwrite parameter is set to True, the INSERT OVERWRITE statement is used.
 - If the boolean overwrite parameter is set to False, the INSERT INTO statement is used.

Descriptions of INSERT OVERWRITE and INSERT INTO:

- INSERT INTO: Upload sessions of the same table or partition do not affect each other. Data uploaded in each session is saved in different directories.
- INSERT OVERWRITE: All data in a table or partition is overwritten by the data in the current upload session. If you use this statement to upload data, do not perform concurrent operations on the same table or partition.

Implementation process

1. The RecordWriter.write() method uploads your data as files to a temporary directory.
2. The RecordWriter.close() method moves the files from the temporary directory to a data directory.
3. The session.commit() method moves all files from the data directory to the directory in which the required table is saved, and updates the table metadata. This way, the data moved to a table in the current job is visible to other MaxCompute jobs such as SQL and MapReduce jobs.

Limits

- The value of a block ID must be greater than or equal to 0 but less than 20000. The size of the data that can be uploaded in a block cannot exceed 100 GB.
- A session is uniquely identified by its ID. The lifecycle of a session is 24 hours. If your session times out because large amounts of data are transmitted, you must transmit your data in multiple sessions.
- The lifecycle of an HTTP request that corresponds to a RecordWriter is 120 seconds. If no data flows over an HTTP connection within 120 seconds, the server closes the connection.

 **Note** HTTP has an 8 KB buffer. When you call the RecordWriter.write() method, your data may be saved to the buffer and no inbound traffic flows over the HTTP connection. In this case, you can call the TunnelRecordWriter.flush() method to forcibly flush data out of the buffer.

- If you use a RecordWriter to write logs to MaxCompute, the write operation may time out due to unexpected traffic fluctuations. To avoid such issues, take note of the following points:
 - We recommend that you do not use a RecordWriter for each data record. If you use a RecordWriter for each data record, a large number of small files are generated, because each RecordWriter corresponds to a file. This affects the performance of MaxCompute.

- If the size of cached code reaches 64 MB, we recommend that you use a RecordWriter to write multiple data records at the same time.
- The lifecycle of a RecordReader is 300 seconds.

7.4.3. InstanceTunnel

This topic describes the InstanceTunnel API.

InstanceTunnel is an entry class to access the MaxCompute Tunnel service. You can use InstanceTunnel to download the execution results of an SQL instance that executes a SELECT statement.

Definition

The following code shows the definition of the InstanceTunnel API:

```
public class InstanceTunnel{
    public DownloadSession createDownloadSession(String projectName, String instanceID);
    public DownloadSession createDownloadSession(String projectName, String instanceID, boolean limitEnabled);
    public DownloadSession getDownloadSession(String projectName, String id);
}
```

Parameters:

- projectName: the name of a project.
- instanceID: the ID of an instance.

Limits

InstanceTunnel provides an easy way to obtain instance execution results. However, it is subject to the following permission limits to ensure data security:

- If the number of data records is less than or equal to 10,000, all users who have read permissions on the specified instance can use InstanceTunnel to download the data records. The same rule applies to data queries by calling a RESTful API.
- If the number of data records is greater than 10,000, only users who have the read permissions on all the source tables from which the specified instance queries data can use InstanceTunnel to download the data records.

7.4.4. UploadSession

This topic describes the UploadSession API.

This API is used to upload data to MaxCompute tables.

Definition

The following code defines the UploadSession API:

```

public class UploadSession {
    UploadSession(Configuration conf, String projectName, String tableName,
                  String partitionSpec) throws TunnelException;
    UploadSession(Configuration conf, String projectName, String tableName,
                  String partitionSpec, String uploadId) throws TunnelException;
    public void commit(Long[] blocks);
    public Long[] getBlockList();
    public String getId();
    public TableSchema getSchema();
    public UploadSession.Status getStatus();
    public Record newRecord();
    public RecordWriter openRecordWriter(long blockId);
    public RecordWriter openRecordWriter(long blockId, boolean compress);
    public RecordWriter openBufferedWriter();
    public RecordWriter openBufferedWriter(boolean compress);
}

```

Notice

- Block IDs that are used within the same upload session must be unique. After you use a block ID to enable RecordWriter, write multiple data records at the same time, call close, and then call commit to complete data upload in an upload session, you cannot use this block ID to enable another RecordWriter to write data.
- The maximum size of a block is 100 GB. We recommend that the volume of data written to each block be greater than 64 MB. Otherwise, the computing performance deteriorates significantly.
- The lifecycle of a session on the server is 24 hours.
- When you upload data, a network action is triggered each time RecordWriter writes 8 KB of data. If no network actions are triggered within 120 seconds, the server closes the connection and RecordWriter becomes unavailable. You must enable a new RecordWriter to write data.
- We recommend that you use the openBufferedWriter operation to upload data. This operation does not show the blockId value but contains an internal data cache. If a block fails to be uploaded, this operation automatically retries the upload process.
- The overwrite mode is added by using the commit method. You can use the overwrite mode to submit data. If you submit data in overwrite mode, the submitted data overwrites the existing data in the table or partition.

 **Notice** Undefined behavior occurs when you submit data in overwrite mode in multiple concurrent sessions. This may affect data accuracy. To avoid this issue, you must determine the number of concurrent sessions in which you submit data in overwrite mode.

Description

- **Lifecycle:** indicates the period from the time an upload instance is created to the time data is uploaded.
- **Upload instance.** You can call the Constructor method or use TableTunnel to create an upload instance.
 - Request mode: synchronous.
 - The server creates a session for the upload instance and generates a unique upload ID to identify the upload instance. You can run the `getId` command on the client to obtain the upload ID.
- **Data upload**
 - Request mode: synchronous.

- Call the `openBufferedWriter` operation to generate a `RecordWriter` instance. The `blockId` parameter identifies the data that is uploaded this time and describes the data position in the whole table. The value of `blockId` is in the range of `[0,20000]`. If the upload fails, you can upload the data again based on the block ID.
- Upload status
 - Request mode: synchronous.
 - Call the `getStatus` method to obtain the current upload status.
 - Call the `getBlockList` method to obtain the blocks that are uploaded. Compare the result with the list of block IDs that were previously sent to the server and re-upload the blocks that fail to be uploaded.
- Upload termination
 - Request mode: synchronous.
 - Call the `Commit(Long[] blocks)` method. The `blocks` parameter indicates the blocks that are uploaded. The server verifies the block list.
 - Verification enhances data accuracy. If the provided list of block IDs is different from the list on the server, an error is returned.
 - If the commit operation fails, try again.
- State description
 - UNKNOWN: This is the initial state when the server creates a session.
 - NORMAL: The upload session is created.
 - CLOSING: When you call the `complete` method to end an upload session, the server changes the state to CLOSING.
 - CLOSED: The data upload is complete. The data is moved to the directory where the result table is saved.
 - EXPIRED: The upload session times out.
 - CRITICAL: An error occurs.

7.4.5. DownloadSession

This topic describes the `DownloadSession` API.

This API is used to download data from MaxCompute tables.

Definition

The following code defines the `DownloadSession` API:

```
public class DownloadSession {
    DownloadSession(Configuration conf, String projectName, String tableName,
                    String partitionSpec) throws TunnelException
    DownloadSession(Configuration conf, String projectName, String tableName,
                    String partitionSpec, String downloadId) throws TunnelException
    public String getId()
    public long getRecordCount()
    public TableSchema getSchema()
    public TableTunnel.DownloadStatus getStatus()
    public RecordReader openRecordReader(long start, long count)
    public RecordReader openRecordReader(long start, long count, boolean compress)
}
```

Description

- Lifecycle: indicates the period from the time a download instance is created to the time data is downloaded.
- Download instance: You can call the constructor method or use `TableTunnel` to create a download instance.

- Request mode: synchronous.
- The server creates a session for the download instance and generates a unique download ID to identify the download instance. You can call the `getId` method on the client to obtain the download ID.
- This operation results in high overheads. The server creates indexes for data files. If a large number of data files exists, it takes a long time to create indexes for the data files.
- The server returns the total number of records. You can start multiple download sessions at the same time to download data based on the total number of data records.
- Data download
 - Request mode: asynchronous.
 - Call the `openRecordReader` API to generate a `RecordReader` instance. The `start` parameter identifies the start position of the data record in this download session. The value of this parameter starts from 0 and must be greater than or equal to 0. The `count` parameter identifies the number of data records that are downloaded in this session. The value of the `count` parameter must be greater than 0.
- Download status
 - Request mode: synchronous.
 - Call the `getStatus` method to obtain the download status.
- Status description
 - UNKNOWN: This is the initial state when the server creates a download session.
 - NORMAL: The download object is created.
 - CLOSED: The download is complete.
 - EXPIRED: The download session times out.

7.4.6. TunnelBufferedWriter

This topic describes the `TunnelBufferedWriter` API.

This API is used to upload data.

The upload process is complex due to limits on block management and connection timeout on the server. The Tunnel SDK provides an enhanced `RecordWriter` of `TunnelBufferWriter` to simplify the upload process.

Definition

The following code defines the `TunnelBufferedWriter` API:

```
public class TunnelBufferedWriter implements RecordWriter {
    public TunnelBufferedWriter(TableTunnel.UploadSession session, CompressOption option) throws IOException;
    public long getTotalBytes();
    public void setBufferSize(long bufferSize);
    public void setRetryStrategy(RetryStrategy strategy);
    public void write(Record r) throws IOException;
    public void close() throws IOException;
}
```

Description

- Lifecycle: indicates the period from the time `RecordWriter` is created to the time data is upload.
- `TunnelBufferedWriter` instance: You can call `openBufferedWriter` of `UploadSession` to create a `TunnelBufferedWriter` instance.
- Data upload: If you call `write`, data records are first written to the local cache. After the cache is full, multiple data records are submitted to the server at a time to avoid a connection timeout. If data upload fails, the

system automatically retries the upload operation.

- Upload termination: You can call `close` and then `commit` of `UploadSession` to terminate the upload process.
- Buffer control: You can call `setBufferSize` to change the memory (in bytes) occupied by the buffer. We recommend that you set the memory size to a value greater than or equal to 64 MB. This prevents excessive small files from being generated on the server, which may affect the processing performance. The value ranges from 1 MB to 1000 MB. The default value is 64 MB.
- Retry policy settings: The following policies are provided: `EXPONENTIAL_BACKOFF`, `LINEAR_BACKOFF`, and `CONSTANT_BACKOFF`. The following code snippet sets the number of Write retries to 6. To avoid unnecessary retries, you can perform each retry after an exponential interval, such as 4s, 8s, 16s, 32s, 64s, and 128s. By default, the interval starts from 4s.

```
RetryStrategy retry
    = new RetryStrategy(6, 4, RetryStrategy.BackoffStrategy.EXPONENTIAL_BACKOFF)
writer = (TunnelBufferedWriter) uploadSession.openBufferedWriter();
writer.setRetryStrategy(retry);
```

 **Note** We recommend that you retain the preceding settings.

7.5. Tunnel SDK example

7.5.1. Example of simple uploads

This topic provides an example of simple uploads.

Example:

```
import java.io.IOException;
import java.util.Date;
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TunnelException;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
public class UploadSample {
    private static String accessId = "<your access id>";
    private static String accessKey = "<your access Key>";
    private static String tunnelUrl = "<your tunnel endpoint>";
    private static String odpsUrl = "<your odps endpoint>";
    private static String project = "<your project>";
    private static String table = "<your table name>";
    private static String partition = "<your partition spec>";
    public static void main(String args[]) {
        Account account = new AliyunAccount(accessId, accessKey);
        Odps odps = new Odps(account);
        odps.setEndpoint(odpsUrl);
        odps.setDefaultProject(project);
        try {
            TableTunnel tunnel = new TableTunnel(odps);
            tunnel.setEndpoint(tunnelUrl);
            PartitionSpec partitionSpec = new PartitionSpec(partition);
            UploadSession uploadSession = tunnel.createUploadSession(project,

```

```

uploadSession uploadSession = tunnel.createUploadSession(project,
    table, partitionSpec);
System.out.println("Session Status is : "
    + uploadSession.getStatus().toString());
TableSchema schema = uploadSession.getSchema();
// After data is prepared, open a writer to start writing data to a block.
// If you write a small amount of data to a block, a large number of small files are generated
. This severely deteriorates computing performance. We strongly recommend that you write more than 6
4 MB of data to a block at a time. The amount of data that can be written to the same block must be
less than 100 GB.
// You can calculate the total amount of data that is written to a block by using the followin
g formula: Total amount of data = Average amount of data in each write operation × Number of write o
perations. The total amount of data must be greater than 64 MB, but less than 100 GB.
RecordWriter recordWriter = uploadSession.openRecordWriter(0);
Record record = uploadSession.newRecord();
for (int i = 0; i < schema.getColumns().size(); i++) {
    Column column = schema.getColumn(i);
    switch (column.getType()) {
        case BIGINT:
            record.setBigint(i, 1L);
            break;
        case BOOLEAN:
            record.setBoolean(i, true);
            break;
        case DATETIME:
            record.setDatetime(i, new Date());
            break;
        case DOUBLE:
            record.setDouble(i, 0.0);
            break;
        case STRING:
            record.setString(i, "sample");
            break;
        default:
            throw new RuntimeException("Unknown column type: "
                + column.getType());
    }
}
for (int i = 0; i < 10; i++) {
    // Write data to the server. A network transmission process is triggered each time 8 KB of d
ata is written.
    // If no network transmission is performed within 120s, the server closes the connection. In
this case, the writer becomes unavailable and you must rewrite data.
    recordWriter.write(record);
}
recordWriter.close();
uploadSession.commit(new Long[] {0L});
System.out.println("upload success!");
} catch (TunnelException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

7.5.2. Example of simple downloads

This topic provides an example of simple downloads.

Example:

```
import java.io.IOException; import java.util.Date;
import com.aliyun.odps.Column; import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec; import com.aliyun.odps.TableSchema; import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount; import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordReader; import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession; import com.aliyun.odps.tunnel.TunnelException;

public class DownloadSample {
private static String accessId = "<your access id>"; private static String accessKey = "<your access Key>";
private static String tunnelUrl = "<your tunnel endpoint>";
private static String odpsUrl = "<your odps endpoint>";
private static String project = "<your project>"; private static String table = "<your table name>";
private static String partition = "<your partition spec>";
public static void main(String args[]) {
Account account = new AliyunAccount(accessId, accessKey); Odps odps = new Odps(account); odps.setEndpoint(odpsUrl);
odps.setDefaultProject(project);
TableTunnel tunnel = new TableTunnel(odps); tunnel.setEndpoint(tunnelUrl);
PartitionSpec partitionSpec = new PartitionSpec(partition); try {
DownloadSession downloadSession = tunnel.createDownloadSession(project, table, partitionSpec);
System.out.println("Session Status is : "
+ downloadSession.getStatus().toString());
long count = downloadSession.getRecordCount(); System.out.println("RecordCount is: " + count);
RecordReader recordReader = downloadSession.openRecordReader(0, count);
Record record;
while ((record = recordReader.read()) != null) { consumeRecord(record, downloadSession.getSchema());
}
recordReader.close();
} catch (TunnelException e) { e.printStackTrace();
} catch (IOException e1) { e1.printStackTrace();
}
}

private static void consumeRecord(Record record, TableSchema schema) { for (int i = 0; i < schema.getColumns().size(); i++) {
Column column = schema.getColumn(i); String colValue = null;
switch (column.getType()) { case BIGINT: {
Long v = record.getBigint(i);
colValue = v == null ? null : v.toString(); break;
}
case BOOLEAN: {
Boolean v = record.getBoolean(i); colValue = v == null ? null : v.toString(); break;
}
case DATETIME: {
Date v = record.getDatetime(i); colValue = v == null ? null : v.toString(); break;
}
case DOUBLE: {
Double v = record.getDouble(i); colValue = v == null ? null : v.toString(); break;
}
case STRING: {
String v = record.getString(i);
colValue = v == null ? null : v.toString(); break;
}
default:
```

```
        }
        throw new RuntimeException("Unknown column type: "
+ column.getType());
    }
    System.out.print(colValue == null ? "null" : colValue); if (i != schema.getColumns().size())
    System.out.print("\t");
    }
    System.out.println();
    }
    }
```

7.5.3. Example of multi-thread uploads

This topic provides an example of multi-thread uploads.

Example:

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TunnelException;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
class UploadThread implements Callable<Boolean> {
    private long id;
    private RecordWriter recordWriter;
    private Record record;
    private TableSchema tableSchema;
    public UploadThread(long id, RecordWriter recordWriter, Record record, TableSchema tableSchema) {
        this.id = id;
        this.recordWriter = recordWriter;
        this.record = record;
        this.tableSchema = tableSchema;
    }
    @Override
    public Boolean call() {
        for (int i = 0; i < tableSchema.getColumns().size(); i++) {
            Column column = tableSchema.getColumn(i);
            switch (column.getType()) {
                case BIGINT:
                    record.setBigint(i, 1L);
                    break;
                case BOOLEAN:
                    record.setBoolean(i, true);
                    break;
                case DATETIME:
                    record.setDatetime(i, new Date());
                    break;
            }
        }
    }
}
```

```

        case DOUBLE:
            record.setDouble(i, 0.0);
            break;
        case STRING:
            record.setString(i, "sample");
            break;
        default:
            throw new RuntimeException("Unknown column type: "
                + column.getType());
    }
}
try {
    for (int i = 0; i < 10; i++) {
        // Write data to the server. A network transmission process is triggered each time 8 KB of data is written.
        // If no network transmission is performed within 120s, the server closes the connection. In this case, the writer becomes unavailable and you must rewrite data.
        recordWriter.write(record);
    }
    recordWriter.close();
} catch (IOException e) {
    e.printStackTrace();
    return false;
}
return true;
}
}

public class UploadThreadSample {
    private static String accessId = "<your access id>";
    private static String accessKey = "<your access Key>";
    private static String tunnelUrl = "<your tunnel endpoint>";
    private static String odpsUrl = "<your odps endpoint>";
    private static String project = "<your project>";
    private static String table = "<your table name>";
    private static String partition = "<your partition spec>";
    private static int threadNum = 10;

    public static void main(String args[]) {
        Account account = new AliyunAccount(accessId, accessKey);
        Odps odps = new Odps(account);
        odps.setEndpoint(odpsUrl);
        odps.setDefaultProject(project);
        try {
            TableTunnel tunnel = new TableTunnel(odps);
            tunnel.setEndpoint(tunnelUrl);
            PartitionSpec partitionSpec = new PartitionSpec(partition);
            UploadSession uploadSession = tunnel.createUploadSession(project,
                table, partitionSpec);
            System.out.println("Session Status is : "
                + uploadSession.getStatus().toString());
            ExecutorService pool = Executors.newFixedThreadPool(threadNum);
            ArrayList<Callable<Boolean>> callers = new ArrayList<Callable<Boolean>>();
            // After data is prepared, open a writer to start writing data in multi-thread mode to a block
            .
            // If you write a small amount of data to a block, a large number of small files are generated
            . This severely deteriorates computing performance. We strongly recommend that you write more than 6
            4 MB of data to a block at a time. The amount of data that can be written to the same block must be
            less than 100 GB.
            // You can calculate the total amount of data that is written to a block by using the following
            q formula: Total amount of data = Average amount of data in each write operation × Number of write o

```

```

perations. The total amount of data must be greater than 64 MB, but less than 100 GB.
    for (int i = 0; i < threadNum; i++) {
        RecordWriter recordWriter = uploadSession.openRecordWriter(i);
        Record record = uploadSession.newRecord();
        callers.add(new UploadThread(i, recordWriter, record, uploadSession.getSchema()));
    }
    pool.invokeAll(callers);
    pool.shutdown();
    Long[] blockList = new Long[threadNum];
    for (int i = 0; i < threadNum; i++)
        blockList[i] = Long.valueOf(i);
    uploadSession.commit(blockList);
    System.out.println("upload success!");
} catch (TunnelException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
}

```

7.5.4. Example of multi-thread downloads

This topic provides an example of multi-thread downloads.

Example:

```

import java.io.IOException;
import java.util.ArrayList; import java.util.Date; import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException; import java.util.concurrent.ExecutorService; import
java.util.concurrent.Executors;
import java.util.concurrent.Future;
import com.aliyun.odps.Column; import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec; import com.aliyun.odps.TableSchema; import com.aliyun.odps.acc
ount.Account;
import com.aliyun.odps.account.AliyunAccount; import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordReader; import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession; import com.aliyun.odps.tunnel.TunnelExcep
tion;
class DownloadThread implements Callable<Long> { private long id;
private RecordReader recordReader; private TableSchema tableSchema;
public DownloadThread(int id,
RecordReader recordReader, TableSchema tableSchema) { this.id = id;
this.recordReader = recordReader; this.tableSchema = tableSchema;
}
@Override
public Long call() {
Long recordNum = 0L; try {
Record record;
while ((record = recordReader.read()) != null) { recordNum++;
System.out.print("Thread " + id + "\t"); consumeRecord(record, tableSchema);
}
recordReader.close();
} catch (IOException e) { e.printStackTrace();
}
}

```

```

return recordNum;
}
private static void consumeRecord(Record record, TableSchema schema) { for (int i = 0; i < schema.getColumns().size(); i++) {
Column column = schema.getColumn(i); String colValue = null;
switch (column.getType()) { case BIGINT: {
Long v = record.getBigint(i);
colValue = v == null ? null : v.toString(); break;
}
case BOOLEAN: {
Boolean v = record.getBoolean(i); colValue = v == null ? null : v.toString(); break;
}
case DATETIME: {
Date v = record.getDatetime(i); colValue = v == null ? null : v.toString(); break;
}
case DOUBLE: {
Double v = record.getDouble(i); colValue = v == null ? null : v.toString(); break;
}
case STRING: {
String v = record.getString(i);
colValue = v == null ? null : v.toString(); break;
}
default:
throw new RuntimeException("Unknown column type: "
+ column.getType());
}
System.out.print(colValue == null ? "null" : colValue); if (i != schema.getColumns().size())
System.out.print("\t");
}
System.out.println();
}
}

public class DownloadThreadSample {
private static String accessId = "<your access id>"; private static String accessKey = "<your access Key>";
private static String tunnelUrl = "<your tunnel endpoint>";
private static String odpsUrl = "<your odps endpoint>";
private static String project = "<your project>"; private static String table = "<your table name>";
private static String partition = "<your partition spec>";
private static int threadNum = 10; public static void main(String args[]) {
Account account = new AliyunAccount(accessId, accessKey);
Odps odps = new Odps(account); odps.setEndpoint(odpsUrl); odps.setDefaultProject(project);
TableTunnel tunnel = new TableTunnel(odps); tunnel.setEndpoint(tunnelUrl);
PartitionSpec partitionSpec = new PartitionSpec(partition); DownloadSession downloadSession;
try {
downloadSession = tunnel.createDownloadSession(project, table, partitionSpec);
System.out.println("Session Status is : "
+ downloadSession.getStatus().toString());
long count = downloadSession.getRecordCount(); System.out.println("RecordCount is: " + count);
ExecutorService pool = Executors.newFixedThreadPool(threadNum); ArrayList<Callable<Long>> callers =
new ArrayList<Callable<Long>>();
long start = 0;
long step = count / threadNum;
for (int i = 0; i < threadNum - 1; i++) {
RecordReader recordReader = downloadSession.openRecordReader( step * i, step);
callers.add(new DownloadThread( i, recordReader, downloadSession.getSchema()));
}
RecordReader recordReader = downloadSession.openRecordReader(step * (threadNum - 1), count
+ ((threadNum - 1) * step));
}
}
}

```

```
callers.add(new DownloadThread( threadNum - 1, recordReader, downloadSession.getSchema()));
Long downloadNum = 0L;
List<Future<Long>> recordNum = pool.invokeAll(callers); for (Future<Long> num : recordNum)
downloadNum += num.get(); System.out.println("Record Count is: " + downloadNum); pool.shutdown();
} catch (TunnelException e) { e.printStackTrace();
} catch (IOException e) { e.printStackTrace();
} catch (InterruptedException e) { e.printStackTrace();
} catch (ExecutionException e) { e.printStackTrace();
}
}
}
```

7.5.5. Example of uploading data by using BufferedWriter

This topic provides an example on how to upload data by using `BufferedWriter` of the MaxCompute Tunnel SDK.

Example:

```
// Initialize the code of MaxCompute and MaxCompute Tunnel.
RecordWriter writer = null;
TableTunnel.UploadSession uploadSession = tunnel.createUploadSession(projectName, tableName);
try {
    int i = 0;
    // Construct a BufferedWriter for MaxCompute Tunnel SDK.
    writer = uploadSession.openBufferedWriter();
    Record product = uploadSession.newRecord();
    for (String item : items) {
        product.setString("name", item);
        product.setBigint("id", i);
        // Call the write() method of BufferedWriter to write data.
        writer.write(product);
        i += 1;
    }
} finally {
    if (writer != null) {
        // Close BufferedWriter of MaxCompute Tunnel SDK.
        writer.close();
    }
}
// Commit the upload session to end the data upload.
uploadSession.commit();
```

7.5.6. Example of uploading data by using BufferedWriter in multithreading mode

This topic provides an example on how to upload data by using `BufferedWriter` of Tunnel SDK in multithreading mode.

Example:

```
class UploadThread extends Thread {
    private UploadSession session;
    private static int RECORD_COUNT = 1200;
    public UploadThread(UploadSession session) {
        this.session = session;
    }
    @Override
    public void run() {
        RecordWriter writer = up.openBufferedWriter();
        Record r = up.newRecord();
        for (int i = 0; i < RECORD_COUNT; i++) {
            r.setBigint(0, i);
            writer.write(r);
        }
        writer.close();
    }
};

public class Example {
    public static void main(String args[]) {
        // Initialize the code of MaxCompute and MaxCompute Tunnel.
        TableTunnel.UploadSession uploadSession = tunnel.createUploadSession(projectName, tableName);
        UploadThread t1 = new UploadThread(up);
        UploadThread t2 = new UploadThread(up);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        uploadSession.commit();
    }
}
```

7.5.7. Examples of uploading and downloading data of complex data types

This topic provides examples of uploading and downloading data of complex data types by using the SDK of MaxCompute Tunnel.

Upload data of complex data types

Example:

```

RecordWriter recordWriter = uploadSession.openRecordWriter(0);
ArrayRecord record = (ArrayRecord) uploadSession.newRecord();
// Prepare data.
List arrayData = Arrays.asList(1, 2, 3);
Map<String, Long> mapData = new HashMap<String, Long>();
mapData.put("a", 1L);
mapData.put("c", 2L);
List<Object> structData = new ArrayList<Object>();
structData.add("Lily");
structData.add(18);
// Pass data into a record.
record.setArray(0, arrayData);
record.setMap(1, mapData);
record.setStruct(2, new SimpleStruct((StructTypeInfo) schema.getColumn(2).getTypeInfo(), structData));
// Write the record to RecordWriter.
recordWriter.write(record);

```

Download data of complex data types

Example:

```

RecordReader recordReader = downloadSession.openRecordReader(0, 1);
// Read a record.
ArrayRecord record1 = (ArrayRecord) recordReader.read();
// Obtain data of the ARRAY type.
List field0 = record1.getArray(0);
List<Long> longField0 = record1.getArray(Long.class, 0);
// Obtain data of the MAP type.
Map field1 = record1.getMap(1);
Map<String, Long> typedField1 = record1.getMap(String.class, Long.class, 1);
// Obtain data of the STRUCT type.
Struct field2 = record1.getStruct(2);

```

Upload and download data of complex data types

Example:

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.RecordReader;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.data.SimpleStruct;
import com.aliyun.odps.data.Struct;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession;
import com.aliyun.odps.tunnel.TunnelException;

```

```

import com.aliyun.odps.type.StructTypeInfo;
public class TunnelComplexTypeSample {
    private static String accessId = "<your access id>";
    private static String accessKey = "<your access Key>";
    private static String odpsUrl = "<your odps endpoint>";
    private static String project = "<your project>";
    private static String table = "<your table name>";
    // Partition in a partitioned table, such as "pt='1',ds='2'".
    // If the table is not a partitioned table, you do not need to execute the following statement.
    private static String partition = "<your partition spec>";
    public static void main(String args[]) {
        Account account = new AliyunAccount(accessId, accessKey);
        Odps odps = new Odps(account);
        odps.setEndpoint(odpsUrl);
        odps.setDefaultProject(project);
        try {
            TableTunnel tunnel = new TableTunnel(odps);
            PartitionSpec partitionSpec = new PartitionSpec(partition);
            //----- Upload data -----
            // Create an upload session for the table.
            // The table schema is {"col0": ARRAY<BIGINT>, "col1": MAP<STRING, BIGINT>, "col2": STRUCT<name:STRING, age:BIGINT>}.
            UploadSession uploadSession = tunnel.createUploadSession(project, table, partitionSpec);
            // Obtain the table schema.
            TableSchema schema = uploadSession.getSchema();
            // Open RecordWriter.
            RecordWriter recordWriter = uploadSession.openRecordWriter(0);
            ArrayRecord record = (ArrayRecord) uploadSession.newRecord();
            // Prepare data.
            List arrayData = Arrays.asList(1, 2, 3);
            Map<String, Long> mapData = new HashMap<String, Long>();
            mapData.put("a", 1L);
            mapData.put("c", 2L);
            List<Object> structData = new ArrayList<Object>();
            structData.add("Lily");
            structData.add(18);
            // Pass data into a record.
            record.setArray(0, arrayData);
            record.setMap(1, mapData);
            record.setStruct(2, new SimpleStruct((StructTypeInfo) schema.getColumn(2).getTypeInfo(), structData));
            // Write the record to RecordWriter.
            recordWriter.write(record);
            // Close RecordWriter.
            recordWriter.close();
            // Commit the upload session to complete the upload.
            uploadSession.commit(new Long[]{0L});
            System.out.println("upload success!");
            //----- Download data -----
            // Create a download session for the table.
            // The table schema is {"col0": ARRAY<BIGINT>, "col1": MAP<STRING, BIGINT>, "col2": STRUCT<name:STRING, age:BIGINT>}.
            DownloadSession downloadSession = tunnel.createDownloadSession(project, table, partitionSpec);
            schema = downloadSession.getSchema();
            // Open RecordReader.
            RecordReader recordReader = downloadSession.openRecordReader(0, 1);
            // Use RecordReader to read a record.
            ArrayRecord record1 = (ArrayRecord) recordReader.read();
            // Obtain data of the ARRAY type.

```

```
List field0 = record1.getArray(0);
List<Long> longField0 = record1.getArray(Long.class, 0);
// Obtain data of the MAP type.
Map field1 = record1.getMap(1);
Map<String, Long> typedField1 = record1.getMap(String.class, Long.class, 1);
// Obtain data of the STRUCT type.
Struct field2 = record1.getStruct(2);
System.out.println("download success!");
} catch (TunnelException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
```

7.6. Appendix

7.6.1. FAQ related to data upload and download by using MaxCompute Tunnel

This topic provides answers to some frequently asked questions about data upload and download by using MaxCompute Tunnel.

What is MaxCompute Tunnel?

MaxCompute Tunnel is a data tunnel that is used to upload data to or download data from MaxCompute. View data cannot be uploaded or downloaded by using MaxCompute Tunnel.

Are duplicate block IDs allowed in an upload session?

No, each block ID in an upload session must be unique. If you use a block ID to open a RecordWriter, write data, and then call the close and commit methods in an upload session, you cannot use this block ID to open another RecordWriter. Valid values of block IDs: [0, 20000).

What is the maximum size of a block?

The maximum size of a block is 100 GB. We recommend that data of greater than 64 MB be written to each block. Each block corresponds to a file. We recommend that you write at least 64 MB of data to each block.

Can a session be shared among processes or threads? What is the lifecycle of a session?

Each session has a 24-hour lifecycle on the server. A session can be used within the 24 hours after it is created, and can be shared among processes or threads. The block ID of each session must be unique. You can perform the following steps to upload data:

1. Create a session.
2. Estimate the amount of data.
3. Assign blocks to threads, for example, assign blocks with the IDs of 0 to 100 to Thread 1 and assign blocks with the IDs of 100 to 200 to Thread 2.
4. Prepare data.
5. Upload data.
6. Commit all blocks to which data is written.

What do I do if read or write operations time out or an I/O exception occurs?

If you upload data, a network action is triggered each time the RecordWriter writes data of 8 KB. If no network action is triggered within 120 seconds, the server closes the connection, and the RecordWriter stops writing data. You must start another RecordWriter to write data.

If you download data, the RecordReader works in the similar way as the RecordWriter. If no network I/O occurs for a long period of time, the server closes the connection. We recommend that you perform read operations without calling other system API operations.

Which SDKs are supported by MaxCompute Tunnel?

MaxCompute Tunnel supports SDK for Java and SDK for C++.

Does MaxCompute Tunnel allow multiple clients to upload the same table at the same time?

Yes, MaxCompute Tunnel allows multiple clients to upload the same table at the same time.

Is MaxCompute Tunnel suitable for batch upload or streaming upload?

MaxCompute Tunnel is suitable for batch upload. It is not suitable for streaming upload.

Do I need to create partitions before I upload data by using MaxCompute Tunnel?

Yes, MaxCompute Tunnel does not automatically create partitions.

What is the relationship between Dship and MaxCompute Tunnel?

Dship is a tool that uploads and downloads data by using MaxCompute Tunnel.

Is data uploaded by using MaxCompute Tunnel appended to an existing file or is the data in the file overwritten by the uploaded data?

The uploaded data is appended to the existing file.

What is the routing feature of MaxCompute Tunnel?

The routing feature allows the SDK of MaxCompute Tunnel to obtain the endpoint of MaxCompute Tunnel by setting the MaxCompute endpoint. Therefore, you can run the SDK of MaxCompute Tunnel after you set the MaxCompute endpoint.

When data is uploaded by using MaxCompute Tunnel, what is the appropriate size of data in each block?

The size of data in each block is determined based on a number of factors, such as the network conditions, real-time performance requirements, data usage, and small files in clusters. In most cases, if the amount of data is large and data is continuously uploaded, the data size can be 64 MB to 256 MB. If data is uploaded once a day, the data size can be about 1 GB.

The timeout error message appears when data is downloaded by using MaxCompute Tunnel. Why?

This issue is usually caused by an incorrect endpoint. You can check the endpoint configuration by using different methods, for example, use Telnet to check network connectivity.

The following error message appears when data is downloaded by using MaxCompute Tunnel. Why?

```
You have NO privilege 'odps:Select' on {acs:odps:*:projects/XXX/tables/XXX}. project 'XXX' is protected
```

The reason is that data protection is enabled for the project from which data is downloaded. The data download you performed aims to transfer data from the project to another project. You can perform data download only after the project owner authorizes you to perform this operation.

The following error message appears when data is uploaded by using MaxCompute Tunnel. Why?

```
ErrorCode=FlowExceeded, ErrorMessage=Your flow quota is exceeded. **
```

The maximum number of concurrent upload or download requests is exceeded. By default, the quota for concurrent requests allowed by MaxCompute Tunnel is 2,000. A request counts against the quota after the request is sent and complete. If you encounter similar errors, we recommend that you use one of the following solutions:

- Make the system work in sleep mode and try again after the system awakes.
- Increase the quota for the project. Before you perform this operation, contact the administrator to estimate the traffic pressure.
- Report the issue to the project owner to locate and control the requests that consume most resources.

7.6.2. Common tunnel error codes

This topic describes common tunnel error codes.

The following table describes the common tunnel error codes.

ErrorCode	Cause	Recommended solution
NoSuchPartition	The partition does not exist.	Tunnel commands cannot be used to create partitions. Create partitions before you upload or download data.
InvalidProjectTable	The project name or table name is invalid.	Rename the project or table to ensure that a valid project or table name is used.
NoSuchProject	The specified project does not exist.	Check whether the project name is valid.
NoSuchTable	The specified table does not exist.	Check whether the table name is valid.
StatusConflict	The session expires or has been committed.	Recreate a session.
MalformedDataStream	The data format is invalid.	If the network connection is closed, reconnect it. If the schema is inconsistent with the table schema, make them consistent.
InvalidPartitionSpec	The partition information is invalid.	Check partition information. An example of a valid partition is pt='1',ct='2017'.
InvalidRowRange	The number of rows exceeds the upper limit or is 0.	Check related parameters.

ErrorCode	Cause	Recommended solution
Unauthorized	The account information, such as the AccessKey ID or AccessKey secret, is invalid, or the time gap between the local device and the server is more than 15 minutes.	Configure a valid AccessKey ID or AccessKey secret or make sure that the maximum time gap between the local device and the server is 15 minutes.
DataStoreError	A storage error occurs.	Contact the administrator.
NoPermission	You are not authorized to perform this operation or you have configured an IP address whitelist.	Check whether the granted permissions are correct.
MissingPartitionSpec	The partition information is not specified. If you want to perform operations on a partitioned table, the partition information must be specified.	Specify the partition information.
TableModified	Table data is modified by other tasks during data uploading or downloading.	Recreate a session.
FlowExceeded	The parallelism exceeds the quota.	Check and control parallelism. Before you increase the parallelism, contact the project owner or administrator to evaluate the traffic pressure.
InvalidResourceSpec	The information of the project, table, or partition is inconsistent with that specified for the session.	Check related information and try again later.
MethodNotAllowed	The method to export views is not supported.	Use other methods.
InvalidColumnSpec	The column name is invalid.	Rename the column to ensure that a valid column name is specified for data downloading.
DataVersionConflict	Cross-cluster replication is performed.	Try again later.
InternalServerError	An internal error occurs.	Try again later or contact the administrator.

8. MaxCompute MapReduce

8.1. Overview

8.1.1. MapReduce

This topic describes the MapReduce programming interfaces supported by MaxCompute.

MaxCompute provides the following types of MapReduce programming interfaces:

- MaxCompute MapReduce: MaxCompute native programming interfaces. These interfaces run fast and are easy to develop a program without exposing file systems.
- Extended MaxCompute MapReduce (MR2): an extension of MaxCompute MapReduce. This type of interface supports the logic to schedule complex jobs. The implementation method of these interfaces is the same as that used by the programming interfaces of MaxCompute MapReduce.
- Hadoop-compatible MapReduce: the programming interfaces that are highly compatible with Hadoop MapReduce. This type of interface is not compatible with MR2.

Note

- The basic concepts, job submission, input and output, and resource usage of the three types of interfaces are similar. The only difference lies in SDKs for Java.
- You cannot use MapReduce to read data from or write data to external tables.

Scenarios

MapReduce supports the following scenarios:

- Search: web crawling, inverted index, and PageRank.
- Analysis of web access logs:
 - Analyze and summarize the characteristics of user behavior, such as web browsing and online shopping. The analysis can be used to deliver personalized recommendations.
 - Analyze user access behavior.
- Statistical analysis of texts:
 - Word count and term frequency-inverse document frequency (TFIDF) analysis of popular novels.
 - Statistical analysis of references to academic papers and patent documents.
 - Wikipedia data analysis.
- Mining of large amounts of data: mining of unstructured data, spatio-temporal data, and image data.
- Machine learning: supervised learning, unsupervised learning, and classification algorithms, such as decision trees and support vector machines (SVMs).
- Natural language processing (NLP):
 - Training and forecast based on big data.
 - Construction of a co-occurrence matrix, mining of frequent itemset data, and duplicate document detection based on existing libraries.
- Advertisement recommendations: forecast of the click-through rate (CTR) and conversion rate (CVR).

Procedure

A MapReduce program processes data in two stages in sequence: the map and reduce stages. You can specify the logic to process data in the map and reduce stages. However, the logic must comply with the conventions of the MapReduce framework. The following procedure shows how the MapReduce framework processes data:

1. Before you perform map operations, make sure that input data is partitioned. After partitioning, the input data is divided into equally sized blocks that are called partitions. Each partition is processed as the input of a single map worker.
2. After partitioning, multiple map workers work at the same time. Each map worker reads its respective partition data, computes the data, and exports the result to a reduce worker.

Note To generate data, a map worker must specify a key for each output record. The key determines the reduce worker for which the data record is targeted. Multiple keys may correspond to a single reduce worker. Data records with the same key are sent to the same reduce worker. A single reduce worker may receive data records with different keys.

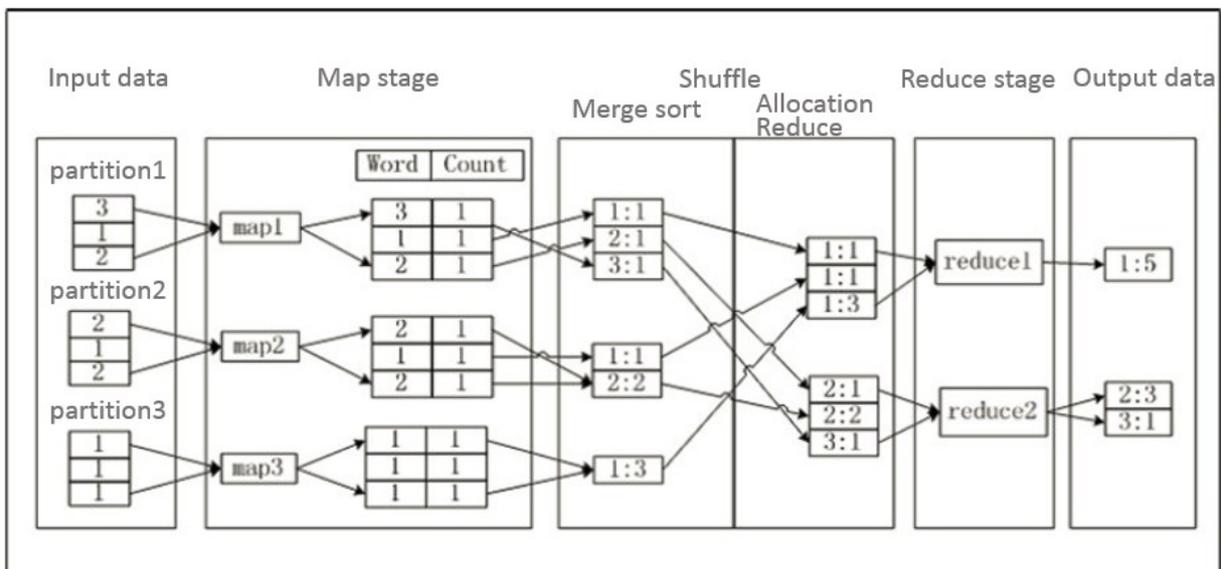
3. Before MapReduce enters the reduce stage, the MapReduce framework sorts data based on key values to make data records with the same key adjacent. If you specify a combiner, the MapReduce framework calls the combiner to combine data records with the same key.

Note You can define the logic of the combiner. Unlike the typical MapReduce framework, MaxCompute requires the input and output parameters of the combiner to be consistent with reduce workers. This process is generally called shuffle.

4. When the MapReduce program enters the Reduce stage, data records with the same key are sent to the same reduce worker. A single reduce worker may receive data records from multiple map workers. Each reduce worker performs the reduce operation on multiple data records with the same key. After the reduce operation, all data records with the same key are converted into a single value.
5. The MapReduce framework generates the result.

The following example describes the concepts of MaxCompute MapReduce for WordCount at different stages.

Assume that a file named a.txt exists and each line of the file contains a digit. You want to count the number of times each digit appears. Each digit is called a word, and the number of times it is used represents the count. The following figure shows how MaxCompute MapReduce counts the words.



1. Partitions the a.txt file and uses data in each partition as the input of a single map worker.
2. For the map processing input, the Count parameter is set to 1 for each obtained word. The <Word, Count> pair is output as a word data key.
3. In the early phase of shuffle, the output of each map worker is sorted by key value (word value). After data records are sorted, the records are combined. To perform this operation, you must accumulate the count values that share the same key value to generate a <Word, Count> pair. This is a sorting and combining

process.

4. In the late phase of shuffle, data is sent to the reduce workers. The reduce workers sort the received data records based on the key values.
5. In the reduce stage, each reduce worker uses the same logic as combiner to process data, and accumulates the count value with the same key value (word value) to obtain the output result.

 **Note** All the MaxCompute data is stored in tables. Therefore, the input and output data of MaxCompute MapReduce can be saved only as tables. You cannot specify the output format, and no interfaces similar to file systems are provided.

8.1.2. MapReduce 2

This topic describes MapReduce 2 (MR2) supported by MaxCompute.

In MapReduce, data must be stored in a MaxCompute table or distributed file system, such as a Hadoop Distributed File System (HDFS), after each round of MapReduce operations. Typically, MapReduce runs multiple MapReduce jobs at the same time. After each job is complete, data is written to disks. However, data may need to be read only once in subsequent map operations to prepare for the shuffle operation. In this case, redundant I/O operations are performed on the disk.

The computing scheduling logic of MaxCompute supports more complicated programming models. Reduce operations can be consecutively performed without having a map operation in between. MaxCompute supports MR2 to allow reduce operations to be consecutively performed after a map operation.

MR2 supported by MaxCompute changes the underlying scheduling and I/O model to avoid redundant I/O operations when you run a job.

Hadoop ChainMapper and ChainReducer also support map or reduce operations in a chained fashion. However, they are essentially different from MR2.

Hadoop ChainMapper and ChainReducer follow the working mechanism used by MapReduce. They allow more than one map operation to be performed after a map or reduce operation. A map or reduce operation cannot be followed by reduce operations. This way, you can reuse the preceding map operation logic to split a map or reduce operation into multiple map stages. This does not change the underlying scheduling or I/O model.

8.1.3. Compatibility with Hadoop MapReduce

This topic describes the background information of compatibility with Hadoop MapReduce. It also describes how to use the Hadoop MapReduce plug-in.

MaxCompute provides a set of programming models and interfaces of Hadoop MapReduce. The input and output of the interfaces are data in MaxCompute tables. The data is organized as records, which demonstrate how the data is processed.

Programming interfaces of MaxCompute MapReduce differ from those of Hadoop MapReduce. To migrate Hadoop MapReduce jobs to MaxCompute MapReduce, you must rewrite the MapReduce code, compile and debug the code by calling MaxCompute MapReduce interfaces, package the final code into a JAR file, and then upload the file to MaxCompute. This process is tedious and labor-intensive for development and testing. It was expected that the original Hadoop MapReduce code can be used on MaxCompute with few modifications or without configurations.

To achieve the ideal solution, MaxCompute provides a plug-in to adapt Hadoop MapReduce to MaxCompute MapReduce. The plug-in enables Hadoop MapReduce jobs to be compatible with MaxCompute MapReduce at the binary level. You can directly run the original Hadoop MapReduce JAR files on MaxCompute after you configure relevant settings. Code rewriting is not required. The plug-in is under testing and does not support custom comparators or key types.

The following procedure describes how to use the Hadoop MapReduce plug-in with the WordCount program.

 Note

- For more information about compatibility with Hadoop MapReduce, see [Compatibility with Hadoop MapReduce](#).
- For more information about the Hadoop MapReduce SDK, see the [MapReduce official documentation](#).

Download the Hadoop MapReduce plug-in

Download the [Hadoop MapReduce](#) plug-in file, `openmr_hadoop2openmr-1.0.jar`.

 **Note** This JAR file contains the dependencies of Hadoop 2.7.2. To avoid version conflicts, you must not include Hadoop dependencies in the JAR files of your jobs.

Prepare the JAR file

Compile and export the WordCount JAR file `wordcount_test.jar`. Sample source code of the WordCount program:

```
package com.aliyun.odps.mapred.example.hadoop;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.util.StringTokenizer;
public class WordCount {
    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
            ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Prepare test data

1. Execute the following statements to create the input table `wc_in` and the output table `wc_out`:

```
create table if not exists wc_in(line string);
create table if not exists wc_out(key string, cnt bigint);
```

2. Prepare the data that you want to import from the `data.txt` file into the `wc_in` table.

Sample data in the `data.txt` file:

```
hello maxcompute
hello mapreduce
```

3. Run the following Tunnel command on the MaxCompute client to import the preceding data from the `data.txt` file to the `wc_in` table:

```
tunnel upload data.txt wc_in;
```

Configure the mapping between HDFS directories and MaxCompute tables

Configure the mapping between HDFS directories and MaxCompute tables in the `wordcount-table-res.conf` file. Sample configurations in the `wordcount-table-res.conf` file:

```
{
  "file:/foo": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver",
      "properties": {
        "text.resolver.columns.combine.enable": "true",
        "text.resolver.seperator": "\t"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_in",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "exact"
  },
  "file:/bar": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.BinaryFileResolver",
      "properties": {
        "binary.resolver.input.key.class" : "org.apache.hadoop.io.Text",
        "binary.resolver.input.value.class" : "org.apache.hadoop.io.LongWritable"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_out",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "fuzzy"
  }
}
```

The `wordcount-table-res.conf` file is a JSON file that describes the mapping between HDFS directories and MaxCompute tables. You must configure the input and output data. Each HDFS directory requires you to configure the following parameters: `resolver`, `tableInfos`, and `matchMode`.

Parameters:

- `resolver`: specifies how to process data in files. The following built-in resolvers can be used: `com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver` and `com.aliyun.odps.mapred.hadoop2openmr.resolver.BinaryFileResolver`. After you specify the resolver parameter, you must configure the required properties for the resolver to parse data.
 - `TextFileResolver`: regards the input or output as plaintext if data is of the plaintext type. If you specify the resolver parameter for input, you must configure the `text.resolver.columns.combine.enable` and `text.resolver.separator` properties. If `text.resolver.columns.combine.enable` is set to true, all columns in the input table are combined into a single string based on the delimiter specified by `text.resolver.separator`. Otherwise, the first two columns in the input table are used to list keys and values.
 - `BinaryFileResolver`: converts binary data into a data type that is supported by MaxCompute, such as `BIGINT`, `BOOLEAN`, and `DOUBLE`. If you specify the resolver parameter for output, you must configure the `binary.resolver.input.key.class` and `binary.resolver.input.value.class` properties. `binary.resolver.input.key.class` specifies the key class of the intermediate result, and `binary.resolver.input.value.class` indicates the value class of the intermediate result.
- `tableInfos`: the MaxCompute table that maps to an HDFS directory. Only the `tblName` parameter can be configured. You must configure the `partSpec` and `label` parameters the same as those in the preceding example.
- `matchMode`: specifies how MaxCompute tables map to HDFS directories. It can be set to `exact` or `fuzzy`. If this parameter is set to `fuzzy`, you can use a regular expression to map tables to HDFS directories.

Submit a job

Run the following command to submit a job on the MaxCompute client, `odpscmd`:

```
jar -DODPS_HADOOPMR_TABLE_RES_CONF=./wordcount-table-res.conf -classpath hadoop2openmr-1.0.jar,wordcount_test.jar com.aliyun.odps.mapred.example.hadoop.WordCount /foo/bar;
```

Note

- `wordcount-table-res.conf`: the configuration file that includes the `wc_in` and `wc_out` tables. The tables are mapped to the `/foo/bar` directory.
- `wordcount_test.jar`: the JAR file of the Hadoop MapReduce plug-in.
- `com.aliyun.odps.mapred.example.hadoop.WordCount`: the class name of the job that you want to run.
- `/foo/bar`: the HDFS directory, which is mapped to the `wc_in` and `wc_out` tables in the JSON configuration file.
- After you configure the mapping, you must use the Data Integration service of DataWorks to import the HDFS input file to the `wc_in` table for MapReduce computing, and export the `wc_out` table to the HDFS output directory `/bar`.
- To run the preceding command, make sure that you have stored the `hadoop2openmr-1.0.jar`, `wordcount_test.jar`, and `wordcount-table-res.conf` files in the `/foo/bar` directory of `odpscmd`. If you do not store the files in the `/foo/bar` directory, modify the configuration and the `-classpath` parameter in the command.

- Each column of the STRING type in a MaxCompute table cannot exceed 8 MB in length.
- If a map or reduce worker does not read or write data, or stops sending heart beats by using `context.progress()`, the default timeout period is 600 seconds.
- When a MapReduce task references a table resource, the supported data types are BIGINT, DOUBLE, STRING, DATETIME, and BOOLEAN. An error is reported for tables of other data types.
- MapReduce does not read OSS data.
- MapReduce does not support the data types that are added to MaxCompute V2.0.

8.3. Features

8.3.1. Run command

This topic describes the command that is used to run MapReduce jobs.

The MaxCompute client provides a jar command for running MapReduce jobs.

Note You must run the following command before you run a MapReduce job:

```
set odps.mr.run.mode=sql;
```

Syntax:

```
Usage: jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS]
-conf <configuration_file> Specify an application configuration file
-classpath <local_file_list> classpaths used to run mainClass
-D <name>=<value> Property value pair, which will be used to run mainClass
-local Run job in local mode
-resources <resource_name_list> file/table resources used in mapper or reducer, separate by comma
```

Parameters:

- `-conf <configuration file>`: the JobConf configuration file.
- `-classpath <local_file_list>`: the classpath that is used to run a MapReduce job in local mode. This parameter specifies the relative and absolute paths of the JAR package where the main function is located.
- `-D <prop_name>=<prop_value>`: the Java property of `<mainClass>` for a MaxCompute job that runs in local mode. You can specify multiple Java properties for a MaxCompute job.
- `-local`: specifies that a MapReduce job runs in local mode. This parameter is used for program debugging.
- `-resources <resource_name_list>`: the resources that are used to run a MapReduce job. You must specify the names of the resources in which the Map or Reduce function is located in `resource_name_list`.

Note If the Map or Reduce function reads data from other MaxCompute resources, you must add the names of these resources to `resource_name_list`. Resource names in `resource_name_list` are separated by commas (.). If you use cross-project resources, add `PROJECT_NAME/resources/` before the resource name, for example, `resources otherproject/resources/resfile`.

You can use the `-conf <configuration file>` parameter to specify the JobConf file. This file contains the settings of JobConf in the SDK. Example of a JobConf file:

```
<configuration>
<property>
<name>import.filename</name>
<value>resource.txt</value>
</property>
</configuration>
```

In the preceding example, the JobConf file defines the `import.filename` variable. The value of this variable is `resource.txt`. You can obtain the value of this variable by calling the JobConf interface in MapReduce. You can also call the JobConf interface in the SDK for the same purpose.

Example:

```
jar -resources mapreduce-examples.jar -classpath mapreduce-examples.jar
org.alidata.odps.mr.examples.WordCount wc_in wc_out
add file data/src.txt
jar -resources src.txt,mapreduce-examples.jar -classpath mapreduce-examples.jar org.alidata.odps.mr.
examples.WordCount wc_in wc_out
add file data/a.txt
add table wc_in as test_table add jar work.jar
jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
-classpath work.jar:otherlib.jar
-D import.filename=resource.txt org.alidata.odps.mr.examples.WordCount args
```

8.3.2. Concepts

8.3.2.1. Map/Reduce

This topic describes Map and Reduce, which are basic concepts in MaxCompute MapReduce.

If a map or reduce task runs, the `setup()`, `map()` or `reduce()`, and `cleanup()` methods are called. The `setup()` method is called before the `map()` or `reduce()` method. Each worker calls the `setup()` method only once. The `cleanup()` method is called after the `map()` or `reduce()` method. Each worker calls the `cleanup()` method only once.

 **Note** For more information about the usage examples, see [Sample programs](#).

8.3.2.2. Sorting

This topic describes sorting, a basic concept in MaxCompute MapReduce.

Some columns in the key records generated by a mapper can be used as sort columns. These columns do not support a custom comparator. You can select a few sort columns as group columns. These columns do not support a custom group comparator. Sort columns are used to sort your data, whereas group columns are used for secondary sorting.

 **Note** For detailed examples, see [Secondary sorting example](#).

8.3.2.3. Partition

This topic describes partition, a basic concept in MaxCompute MapReduce.

MaxCompute allows you to configure partition key columns and custom partitioners. Partition key columns take precedence over custom partitioners. Partitioners are used to allocate data generated by a mapper to different reducers based on the partition logic.

8.3.2.4. Combiner

This topic describes the combiner function, a basic concept in MaxCompute MapReduce.

The combiner function combines adjacent records at the shuffle stage. You can determine whether to use the combiner function based on your business logic.

The combiner function is the optimization of the MapReduce computing framework. The combiner logic is the same as the reducer logic. After a mapper generates data, the framework combines the data with the same key at the map stage.

 **Note** For detailed examples, see [Sample programs](#).

8.3.3. Submit a job

This topic describes how to run a jar command on the MaxCompute client to submit a MapReduce job.

The MaxCompute client provides a jar command to submit MapReduce jobs. Command syntax:

```
jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS];
    -conf <configuration_file>      Specify an application configuration file
    -resources <resource_name_list> file\table resources used in mapper or reducer, separate
    by comma
    -classpath <local_file_list>    classpaths used to run mainClass
    -D <name>=<value>               Property value pair, which will be used to run mainClass
    -l                               Run job in local mode
```

Parameters:

- **-conf <configuration file>**: the JobConf file. This file can affect the settings of JobConf in an SDK.

Template of a JobConf file:

```
<configuration>
  <property>
    <name>import.filename</name>
    <value>resource.txt</value>
  </property>
</configuration>
```

In the preceding template, the JobConf file defines the `import.filename` variable. The value of this variable is `resource.txt`. You can obtain the value of this variable over the JobConf interface in MapReduce. You can also call the JobConf interface in an SDK to obtain the value of the variable. For more information, see [Resource usage example](#).

Example:

```
add jar data\mapreduce-examples.jar;
jar -resources mapreduce-examples.jar -classpath data\mapreduce-examples.jar
org.alidata.odps.mr.examples.WordCount wc_in wc_out;
add file data\src.txt;
add jar data\mapreduce-examples.jar;
jar -resources src.txt,mapreduce-examples.jar -classpath data\mapreduce-examples.jar
org.alidata.odps.mr.examples.WordCount wc_in wc_out;
add file data\a.txt;
add table wc_in as test_table;
add jar data\work.jar;
jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
-classpath data\work.jar:otherlib.jar
-D import.filename=resource.txt org.alidata.odps.mr.examples.WordCount args;
```

- `-resources <resource_name_list>`: the resources that are used to run a MapReduce job. Typically, `resource_name_list` specifies the name of the resource in which the Map or Reduce function is located.

Note If the Map or Reduce function reads data from other MaxCompute resources, you must add the names of these resources to `resource_name_list`. The resource names are separated by commas (,). If you use cross-project resources, add `PROJECT/resources/` before `resource_name_list`, for example, `-resources otherproject/resources/resfile`.

- `-classpath <local_file_list>`: the classpath that is used to run a MapReduce job in local mode. This parameter specifies the relative and absolute paths of the JAR packages that encapsulates the main function. Package names are separated by default file delimiters. In most cases, the Windows operating system uses semicolons (;) as the default file delimiter, and the Linux operating system uses commas (,) as the default file delimiter. If you run a MapReduce job on a cloud server, separate package names with commas (,).

Note The main function and Map/Reduce function are usually encapsulated into the same package. If you run the related program, `mapreduce-examples.jar` is specified in the following parameters: `-resources <resource_name_list>` and `-classpath <local_file_list>`. However, `-resources <resource_name_list>` references the Map or Reduce function and is run in a distributed environment, whereas `-classpath <local_file_list>` references the main function and is run in local mode with the specified JAR package saved in a local directory.

- `-D <name>=<value>`: the Java attribute of `<mainClass>` when you run a MapReduce job in local mode. You can define multiple Java attributes.
- `-l`: specifies that the MapReduce job is executed in local mode. This parameter is used for program debugging.

Example:

```
jar -conf \home\admin\myconf -resources a.txt,example.jar -classpath ..\lib\example.jar:.\other_lib.
jar -D java.library.path=.\native;
```

8.3.4. Inputs and outputs

This topic describes the inputs and outputs of MapReduce jobs in MaxCompute.

- The inputs and outputs of MapReduce jobs in MaxCompute support built-in data types of MaxCompute, including BIGINT, DOUBLE, STRING, DATETIME, and BOOLEAN. User-defined data types are not supported.
- MapReduce supports input data from multiple tables with different schemas. You can use the map function to obtain the table information that corresponds to the current record.
- MapReduce supports null values as input data but does not support views as input data.
- A reduce job can write data to different tables or different partitions of a table. The destination tables or

partitions can have different schemas. You can specify labels to distinguish outputs. If you want to use the default output, you do not need to specify a label. MapReduce does not support a function without output returned.

 **Note** For more information about input and output examples, see [Example: Input and output data to multiple objects](#).

8.3.5. Use resources

This topic describes how MapReduce uses resources.

You can use the Map or Reduce function to read data from MaxCompute resources. A Map or Reduce worker loads resources to the memory for you to write code.

 **Note** For a detailed example, see [Resource usage example](#).

8.3.6. Job running in local mode

This topic describes the differences between the local mode and distributed mode in which MapReduce jobs run. It also provides examples of MapReduce jobs in local mode.

Introduction to the local mode

Before you run a job in local mode, you can specify the `-local` option in the JAR command to simulate the running of the job. This way, you can perform local debugging on the job.

During job running, the client downloads the metadata and data of the input table, metadata of the output table, and resources that are required for local debugging from MaxCompute. The downloaded data is saved to a local directory named *warehouse*.

After the job is completed, the computing results are saved to a file in the *warehouse* directory. If the input table and required resources are downloaded to the *warehouse* directory, MapReduce directly references the data and files in the directory next time, instead of downloading the data again.

Differences between the local mode and distributed mode

A MapReduce job that runs in local mode starts multiple map and reduce tasks to process data. These tasks run in sequence rather than in parallel. The simulated running process is different from an actual distributed running process in the following aspects:

- Rows in the input table: A maximum of 100 rows of data can be downloaded in local mode.
- Resource usage: In distributed mode, MaxCompute limits the size of resources that can be referenced. However, no limits are imposed on the size of resources in local mode.
- Security: MaxCompute MapReduce and user-defined functions (UDFs) are limited by a Java sandbox in distributed mode. However, no limits are imposed in local mode.

Examples

The following code shows an example of a MapReduce job in local mode:

```

odps:my_project> jar -l com.aliyun.odps.mapred.example.WordCount wc_in wc_out
Summary:
counters: 10
  map-reduce framework
    combine_input_groups=2
    combine_output_records=2
    map_input_bytes=4
    map_input_records=1
    map_output_records=2
    map_output_[wc_out]_bytes=0
    map_output_[wc_out]_records=0
    reduce_input_groups=2
    reduce_output_[wc_out]_bytes=8
    reduce_output_[wc_out]_records=2
OK

```

Note For more information about the sample code of WordCount, see [WordCount example](#).

If this is the first time you run a local debugging command, a directory named `warehouse` is created in the current path after the command is executed. The following code shows the directory structure of `warehouse`.

```

<warehouse>
|__my_project (project directory)
|  |__ <__tables__>
|  |  |__wc_in (table data directory)
|  |  |  |__ data (file)
|  |  |  |
|  |  |  |__ <__schema__> (file)
|  |  |__wc_out (table data directory)
|  |  |  |__ data (file)
|  |  |  |
|  |  |  |__ <__schema__> (file)
|  |
|  |__ <__resources__>
|  |  |
|  |  |__table_resource_name (table resource)
|  |  |  |__<__ref__>
|  |  |
|  |  |__file_resource_name (file resource)

```

- Directories at the same level as `my_project` indicate projects. Directories at the same level as `wc_in` and `wc_out` indicate data tables. The table data that you read or write by using the JAR command is downloaded to directories at this level.
- The schema file stores the metadata of a table. The following code defines the file format:

```

project=local_project_name
table=local_table_name
columns=col1_name:col1_type,col2_name:col2_type
partitions=p1:STRING,p2:BIGINT -- In this example, you do not need to specify this field.

```

Note Separate the name and data type of a column with a colon (:). Separate columns with commas (.). The project and table names, `projectname.tablename`, must be declared at the beginning of the schema file. Separate the declaration and column definition with a comma (,), for example, `project_name.table_name,col1_name:col1_type,col2_name:col2_type,.....`

- The data file in the *tables* directory stores the table data. The number of columns and column data must match the definition in the schema file. Separate columns with commas (,).

For example, the schema file in the *wc_in* directory contains the following data:

```
my_project.wc_in,key:STRING,value:STRING
```

In this case, the data file contains the following data:

```
0,2
```

The client downloads the metadata and part of the data of a table from MaxCompute and saves the data to the preceding files. The next time you run this example program, the client directly uses the data in the *wc_in* directory, instead of downloading it again.

 **Note** Data can be downloaded from MaxCompute only for MapReduce jobs that run in local mode.

For example, the schema file in the *wc_out* directory contains the following data:

```
my_project.wc_out,key:STRING,cnt:BIGINT
```

In this case, the data file contains the following data:

```
0,1
2,1
```

The client downloads the metadata of the *wc_out* table from MaxCompute and saves the data to the schema file. After a job is completed, the results are saved to the data file.

 **Note**

- You can also edit the schema and data files and save the files in table directories.
- If you run a job in local mode and the client detects that the table directory exists, the client does not download the information of this table from MaxCompute. The local table directory can include a table that does not exist in MaxCompute.

8.4. SDK introduction

8.4.1. Overview of major MapReduce APIs

This topic describes major MapReduce APIs.

If you use Maven, you can obtain Maven dependencies from the [Maven Central Repository](#).

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-mapred</artifactId>
  <version>0.36.2-public</version>
</dependency>
```

API	Description
-----	-------------

API	Description
MapperBase	The base class that custom Map functions must inherit. A mapper converts the records in the input table to key-value pairs and passes the key-value pairs to a reducer. Alternatively, a mapper can directly write the key-value pairs to a result table by skipping the Reduce stage. The jobs that skip the Reduce stage and directly return computing results are called map-only jobs.
ReducerBase	The base class that custom Reduce functions must inherit. A reducer reduces a set of values that are associated with a key.
TaskContext	Describes the context of a task. The task context is an input parameter of multiple member functions of MapperBase and ReducerBase.
JobClient	Submits and manages jobs. Jobs can be submitted in blocking mode (synchronous) or non-blocking mode (asynchronous).
RunningJob	Defines a running job. The objects of this class are used to track the instances on which MapReduce jobs are running.
JobConf	Describes the configuration of a MapReduce job. The JobConf object is defined in the main function. Then, a job client submits a job to MaxCompute based on the JobConf object.

8.4.2. API description

8.4.2.1. MapperBase

This topic describes the main function interfaces of the MapperBase API.

The following table describes the main function interfaces of MapperBase.

Interface	Description
void cleanup(TaskContext context)	Calls the related method at the end of the map stage after the map method is called.
void map(long key, Record record, TaskContext context)	Calls the map method to process records in an input table.
void setup(TaskContext context)	Calls the related method at the beginning of the map stage before the map method is called.

8.4.2.2. ReducerBase

This topic describes the main function interfaces of the ReducerBase API.

The following table describes the main function interfaces of ReducerBase.

Interface	Description
void cleanup(TaskContext context)	Calls the related method at the end of the reduce stage after the reduce method is called.
void reduce(Record key, Iterator<Record > values, TaskContext context)	Calls the reduce method to process records in an input table.
void setup(TaskContext context)	Calls the related method at the beginning of the reduce stage before the reduce method is called.

8.4.2.3. TaskContext

This topic describes the main function interfaces of the TaskContext API.

The following table describes the main function interfaces of TaskContext.

Interface	Description
TableInfo[] getOutputTableInfo()	Gets output table information.
Record createOutputRecord()	Creates records for the default output table.
Record createOutputRecord(String label)	Creates records for the output table with the specified label.
Record createMapOutputKeyRecord()	Creates records for keys in the key-value pairs that are generated at the map stage.
Record createMapOutputValueRecord()	Creates records for values in the key-value pairs that are generated at the map stage.
void write(Record record)	Writes records to the default output table. The interface can be called multiple times at the reduce stage.
void write(Record record, String label)	Writes records to the output table with the specified label. The interface can be called multiple times at the reduce stage.
void write(Record key, Record value)	Writes records to the intermediate results. The interface can be called multiple times at the map stage.
BufferedInputStream readResourceFileAsStream(String resourceName)	Reads a file resource.
Iterator<Record > readResourceTable(String resourceName)	Reads a table resource.
Counter getCounter(Enum<? > name)	Obtains the counter with the specified name.
Counter getCounter(String group, String name)	Obtains the counter with the specified name in the specified group.
void progress()	Sends heartbeat information to the MapReduce framework. If your task takes an extended period of time to process data and you do not need to call the framework during this time period, you can call this interface to avoid a task timeout. The default timeout period for a task is 600 seconds.

Note

- The TaskContext API has a progress interface, which prevents it from being forced out due to timeout if a worker runs for a long time. This interface sends heartbeats to the framework rather than reporting the worker progress.
- The default timeout period for a worker is 10 minutes in MaxCompute MapReduce. You cannot change the timeout period. If a worker does not send heartbeat information by calling the progress interface in 10 minutes, the framework terminates the worker and the map or reduce task fails. Therefore, we recommend that you periodically call the progress interface in a map or reduce task to prevent the framework from terminating workers unexpectedly.

8.4.2.4. JobConf

This topic describes the main function interfaces of the JobConf API.

The following table describes the main function interfaces of JobConf.

Interface	Description
<code>void setResources(String resourceNames)</code>	Declares the resources that are used in the current job. A mapper or reducer can read only the resources that have been declared in the TaskContext object.
<code>void setMapOutputKeySchema(Column[] schema)</code>	Sets the attributes of keys that are passed from the mapper to the reducer.
<code>void setMapOutputValueSchema(Column[] schema)</code>	Sets the attributes of values that are passed from the mapper to the reducer.
<code>void setOutputKeySortColumns(String[] cols)</code>	Sets the columns for sorting the keys that are passed from the mapper to the reducer.
<code>void setOutputGroupingColumns(String[] cols)</code>	Sets the columns for grouping the keys.
<code>void setMapperClass(Class<? extends Mapper > > theClass)</code>	Sets a mapper for a job.
<code>void setPartitionColumns(String[] cols)</code>	Sets the partition key columns for a job. By default, the partition key columns are all columns of the keys that are generated by the mapper.
<code>void setReducerClass(Class<? extends Reducer > theClass)</code>	Sets a reducer for a job.
<code>void setCombinerClass(Class<? extends Reducer > theClass)</code>	Sets a combiner for a job. A combiner combines records with the same key. It is similar to a reducer but works at the map stage.
<code>void setSplitSize(long size)</code>	Sets the split size. Unit: MB. The default split size is 256 MB.
<code>void setNumReduceTasks(int n)</code>	Sets the number of reduce tasks. By default, the number of reduce tasks is one-fourth of the number of map tasks.
<code>void setMemoryForMapTask(int mem)</code>	Sets the memory available to a worker in a map task. Unit: MB. The default memory size is 2048 MB.

Interface	Description
void setMemoryForReduceTask(int mem)	Sets the memory available to a worker in a reduce task. Unit: MB. The default memory size is 2048 MB.
void setOutputSchema(Column[] schema, String label)	Sets the output attribute of a designated label. If data is inserted into multiple objects, each output corresponds to a label.

 **Note**

- The grouping columns are selected from the sort columns. The sort columns and partition key columns must exist in keys.
- At the map stage, the hash values of records from a mapper are calculated based on the specified partition key columns. The hash values help determine the reducers to which records are passed. The records are sorted based on the sort columns before the records are passed to reducers.
- At the reduce stage, input records are grouped based on the grouping columns. Then, a group of records that share the same key are passed to the reduce method as input.

8.4.2.5. JobClient

This topic describes the main function interfaces of the JobClient API.

The following table describes the main function interfaces of JobClient.

Interface	Description
static RunningJob runJob(JobConf job)	Submits a MapReduce job in blocking (synchronous) mode and returns a RunningJob object.
static RunningJob submitJob(JobConf job)	Submits a MapReduce job in non-blocking (asynchronous) mode and returns a RunningJob object.

8.4.2.6. RunningJob

This topic describes the main function interfaces of the RunningJob API.

The following table describes the main function interfaces of RunningJob.

Interface	Description
String getInstanceID()	Obtains the ID of a job instance. You can use the job instance ID to view operational logs and manage jobs.
boolean isComplete()	Checks whether a job is completed.
boolean isSuccessful()	Checks whether a job instance is successful.
void waitForCompletion()	Waits for a job instance to end. The method is used for jobs that are submitted in asynchronous mode.
JobStatus getJobStatus()	Checks the running status of a job instance.
void killJob()	Ends the current job.

Interface	Description
Counters getCounters()	Obtains the counter information.

8.4.2.7. InputUtils

This topic describes the main function interfaces of the InputUtils API.

The following table describes the main function interfaces of InputUtils.

Interface	Description
static void addTable(TableInfo table, JobConf conf)	Adds an input table to a task. The method can be called multiple times. New tables are appended to the input queue.
static void setTables(TableInfo [] tables, JobConf conf)	Adds multiple input tables to a task.

8.4.2.8. OutputUtils

This topic describes the main function interfaces of the OutputUtils API.

The following table describes the main function interfaces of OutputUtils.

Interface	Description
static void addTable(TableInfo table, JobConf conf)	Adds an output table to a task. The method can be called multiple times. New tables are appended to the output queue.
static void setTables(TableInfo [] tables, JobConf conf)	Adds multiple output tables to a task.

8.4.2.9. Pipeline

This topic describes the main interfaces of the Pipeline API.

Pipeline is the main class of MR2. You can call the Pipeline.builder method to build a pipeline. The following code shows the main interfaces of the Pipeline class:

```
public Builder addMapper(Class<? extends Mapper> mapper)
public Builder addMapper(Class<? extends Mapper> mapper,
    Column[] keySchema, Column[] valueSchema, String[] sortCols,
    SortOrder[] order, String[] partCols,
    Class<? extends Partitioner> theClass, String[] groupCols)
public Builder addReducer(Class<? extends Reducer> reducer)
public Builder addReducer(Class<? extends Reducer> reducer,
    Column[] keySchema, Column[] valueSchema, String[] sortCols,
    SortOrder[] order, String[] partCols,
    Class<? extends Partitioner> theClass, String[] groupCols)
public Builder setOutputKeySchema(Column[] keySchema)
public Builder setOutputValueSchema(Column[] valueSchema)
public Builder setOutputKeySortColumns(String[] sortCols)
public Builder setOutputKeySortOrder(SortOrder[] order)
public Builder setPartitionColumns(String[] partCols)
public Builder setPartitionerClass(Class<? extends Partitioner> theClass)
public Builder setOutputGroupingColumns(String[] cols)
```

The following example shows how to call the Pipeline.builder method to build a pipeline:

```

Job job = new Job();
Pipeline pipeline = Pipeline.builder()
    .addMapper(TokenizerMapper.class)
    .setOutputKeySchema(
        new Column[] { new Column("word", OdpsType.STRING) })
    .setOutputValueSchema(
        new Column[] { new Column("count", OdpsType.BIGINT) })
    .addReducer(SumReducer.class)
    .setOutputKeySchema(
        new Column[] { new Column("count", OdpsType.BIGINT) })
    .setOutputValueSchema(
        new Column[] { new Column("word", OdpsType.STRING),
            new Column("count", OdpsType.BIGINT) })
    .addReducer(IdentityReducer.class).createPipeline();
job.setPipeline(pipeline);
job.addInput(...)
job.addOutput(...)
job.submit();
    
```

As shown in the preceding example, you can create a MapReduce job in which a mapper is followed by two reducers in the Main function. If you are familiar with the basic features of MapReduce, MR2 is easy to use.

Note

- Before you use MR2, we recommend that you learn how to use MapReduce.
- You can create a MapReduce job in which a mapper is followed by only one reducer by using JobConf.

8.4.3. Compatibility with Hadoop MapReduce

This topic describes the compatibility between specific MaxCompute MapReduce interfaces and Hadoop MapReduce.

The following table describes whether specific MaxCompute MapReduce interfaces are compatible with Hadoop MapReduce.

Type	Interface	Compatible with Hadoop MapReduce
Mapper	void map(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context)	Yes.
Mapper	void run(org.apache.hadoop.mapreduce.Mapper.Context context)	Yes.
Mapper	void setup(org.apache.hadoop.mapreduce.Mapper.Context context)	Yes.
Reducer	void cleanup(org.apache.hadoop.mapreduce.Reducer.Context context)	Yes.

Type	Interface	Compatible with Hadoop MapReduce
Reducer	void reduce(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Reducer.Context context)	Yes.
Reducer	void run(org.apache.hadoop.mapreduce.Reducer.Context context)	Yes.
Reducer	void setup(org.apache.hadoop.mapreduce.Reducer.Context context)	Yes.
Partitioner	int getPartition(KEY key, VALUE value, int numPartitions)	Yes.
MapContext, which extends TaskInputOutputContext	InputSplit getInputSplit()	No. An exception is reported.
ReduceContext	nextKey()	Yes.
ReduceContext	getValues()	Yes.
TaskInputOutputContext	getCurrentKey()	Yes.
TaskInputOutputContext	getCurrentValue()	Yes.
TaskInputOutputContext	getOutputCommitter()	No. An exception is reported.
TaskInputOutputContext	nextKeyValue()	Yes.
TaskInputOutputContext	write(KEYOUT key, VALUEOUT value)	Yes.
TaskAttemptContext	getCounter(Enum<?> counterName)	Yes.
TaskAttemptContext	getCounter(String groupName, String counterName)	Yes.
TaskAttemptContext	setStatus(String msg)	Empty implementation.
TaskAttemptContext	getStatus()	Empty implementation.
TaskAttemptContext	getTaskAttemptID()	No. An exception is reported.
TaskAttemptContext	getProgress()	No. An exception is reported.
TaskAttemptContext	progress()	Yes.
Job	addArchiveToClassPath(Path archive)	No.
Job	addCacheArchive(URI uri)	No.
Job	addCacheFile(URI uri)	No.
Job	addFileToClassPath(Path file)	No.
Job	cleanupProgress()	No.

Type	Interface	Compatible with Hadoop MapReduce
Job	createSymlink()	No. An exception is reported.
Job	failTask(TaskAttemptID taskId)	No.
Job	getCompletionPollInterval(Configuration conf)	Empty implementation.
Job	getCounters()	Yes.
Job	getFinishTime()	Yes.
Job	getHistoryUrl()	Yes.
Job	getInstance()	Yes.
Job	getInstance(Cluster ignored)	Yes.
Job	getInstance(Cluster ignored, Configuration conf)	Yes.
Job	getInstance(Configuration conf)	Yes.
Job	getInstance(Configuration conf, String jobName)	Empty implementation.
Job	getInstance(JobStatus status, Configuration conf)	No. An exception is reported.
Job	getJobFile()	No. An exception is reported.
Job	getJobName()	Empty implementation.
Job	getJobState()	No. An exception is reported.
Job	getPriority()	No. An exception is reported.
Job	getProgressPollInterval(Configuration conf)	Empty implementation.
Job	getReservationId()	No. An exception is reported.
Job	getSchedulingInfo()	No. An exception is reported.
Job	getStartTime()	Yes.
Job	getStatus()	No. An exception is reported.
Job	getTaskCompletionEvents(int startFrom)	No. An exception is reported.
Job	getTaskCompletionEvents(int startFrom, int numEvents)	No. An exception is reported.
Job	getTaskDiagnostics(TaskAttemptID taskId)	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce
Job	getTaskOutputFilter(Configuration conf)	No. An exception is reported.
Job	getTaskReports(TaskType type)	No. An exception is reported.
Job	getTrackingURL()	Yes.
Job	isComplete()	Yes.
Job	isRetired()	No. An exception is reported.
Job	isSuccessful()	Yes.
Job	isUber()	Empty implementation.
Job	killJob()	Yes.
Job	killTask(TaskAttemptID taskId)	No.
Job	mapProgress()	Yes.
Job	monitorAndPrintJob()	Yes.
Job	reduceProgress()	Yes.
Job	setCacheArchives(URI[] archives)	No. An exception is reported.
Job	setCacheFiles(URI[] files)	No. An exception is reported.
Job	setCancelDelegationTokenUponJobCompletion(boolean value)	No. An exception is reported.
Job	setCombinerClass(Class<? extends Reducer> cls)	Yes.
Job	setCombinerKeyGroupingComparatorClass(Class<? extends RawComparator> cls)	Yes.
Job	setGroupingComparatorClass(Class<? extends RawComparator> cls)	Yes.
Job	setInputFormatClass(Class<? extends InputFormat> cls)	Empty implementation.
Job	setJar(String jar)	Yes.
Job	setJarByClass(Class<? > cls)	Yes.
Job	setJobName(String name)	Empty implementation.
Job	setJobSetupCleanupNeeded(boolean needed)	Empty implementation.
Job	setMapOutputKeyClass(Class<? > theClass)	Yes.

Type	Interface	Compatible with Hadoop MapReduce
Job	setMapOutputValueClass(Class<? > theClass)	Yes.
Job	setMapperClass(Class<? extends Mapper> cls)	Yes.
Job	setMapSpeculativeExecution(boolean speculativeExecution)	Empty implementation.
Job	setMaxMapAttempts(int n)	Empty implementation.
Job	setMaxReduceAttempts(int n)	Empty implementation.
Job	setNumReduceTasks(int tasks)	Yes.
Job	setOutputFormatClass(Class<? extends OutputFormat> cls)	No. An exception is reported.
Job	setOutputKeyClass(Class<? > theClass)	Yes.
Job	setOutputValueClass(Class<? > theClass)	Yes.
Job	setPartitionerClass(Class<? extends Partitioner> cls)	Yes.
Job	setPriority(JobPriority priority)	No. An exception is reported.
Job	setProfileEnabled(boolean newValue)	Empty implementation.
Job	setProfileParams(String value)	Empty implementation.
Job	setProfileTaskRange(boolean isMap, String newValue)	Empty implementation.
Job	setReducerClass(Class<? extends Reducer> cls)	Yes.
Job	setReduceSpeculativeExecution(boolean speculativeExecution)	Empty implementation.
Job	setReservationId(ReservationId reservationId)	No. An exception is reported.
Job	setSortComparatorClass(Class<? extends RawComparator> cls)	No. An exception is reported.
Job	setSpeculativeExecution(boolean speculativeExecution)	Yes.
Job	setTaskOutputFilter(Configuration conf, org.apache.hadoop.mapreduce.Job.TaskStatusFilter newValue)	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce
Job	setUpProgress()	No. An exception is reported.
Job	setUser(String user)	Empty implementation.
Job	setWorkingDirectory(Path dir)	Empty implementation.
Job	submit()	Yes.
Job	toString()	No. An exception is reported.
Job	waitForCompletion(boolean verbose)	Yes.
Task Execution & Environment	mapreduce.map.java.opts	Empty implementation.
Task Execution & Environment	mapreduce.reduce.java.opts	Empty implementation.
Task Execution & Environment	mapreduce.map.memory.mb	Empty implementation.
Task Execution & Environment	mapreduce.reduce.memory.mb	Empty implementation.
Task Execution & Environment	mapreduce.task.io.sort.mb	Empty implementation.
Task Execution & Environment	mapreduce.map.sort.spill.percent	Empty implementation.
Task Execution & Environment	mapreduce.task.io.soft.factor	Empty implementation.
Task Execution & Environment	mapreduce.reduce.merge.inmem.thresholds	Empty implementation.
Task Execution & Environment	mapreduce.reduce.shuffle.merge.percent	Empty implementation.
Task Execution & Environment	mapreduce.reduce.shuffle.input.buffer.percent	Empty implementation.
Task Execution & Environment	mapreduce.reduce.input.buffer.percent	Empty implementation.
Task Execution & Environment	mapreduce.job.id	Empty implementation.
Task Execution & Environment	mapreduce.job.jar	Empty implementation.
Task Execution & Environment	mapreduce.job.local.dir	Empty implementation.
Task Execution & Environment	mapreduce.task.id	Empty implementation.
Task Execution & Environment	mapreduce.task.attempt.id	Empty implementation.
Task Execution & Environment	mapreduce.task.is.map	Empty implementation.
Task Execution & Environment	mapreduce.task.partition	Empty implementation.
Task Execution & Environment	mapreduce.map.input.file	Empty implementation.
Task Execution & Environment	mapreduce.map.input.start	Empty implementation.

Type	Interface	Compatible with Hadoop MapReduce
Task Execution & Environment	mapreduce.map.input.length	Empty implementation.
Task Execution & Environment	mapreduce.task.output.dir	Empty implementation.
JobClient	cancelDelegationToken(Token <DelegationTokenIdentifier> token)	No. An exception is reported.
JobClient	close()	Empty implementation.
JobClient	displayTasks(JobID jobId, String type, String state)	No. An exception is reported.
JobClient	getAllJobs()	No. An exception is reported.
JobClient	getCleanupTaskReports(JobID jobId)	No. An exception is reported.
JobClient	getClusterStatus()	No. An exception is reported.
JobClient	getClusterStatus(boolean detailed)	No. An exception is reported.
JobClient	getDefaultMaps()	No. An exception is reported.
JobClient	getDefaultReduces()	No. An exception is reported.
JobClient	getDelegationToken(Text renewer)	No. An exception is reported.
JobClient	getFs()	No. An exception is reported.
JobClient	getJob(JobID jobId)	No. An exception is reported.
JobClient	getJob(String jobId)	No. An exception is reported.
JobClient	getJobsFromQueue(String queueName)	No. An exception is reported.
JobClient	getMapTaskReports(JobID jobId)	No. An exception is reported.
JobClient	getMapTaskReports(String jobId)	No. An exception is reported.
JobClient	getQueueAclsForCurrentUser()	No. An exception is reported.
JobClient	getQueueInfo(String queueName)	No. An exception is reported.
JobClient	getQueues()	No. An exception is reported.
JobClient	getReduceTaskReports(JobID jobId)	No. An exception is reported.
JobClient	getReduceTaskReports(String jobId)	No. An exception is reported.
JobClient	getSetupTaskReports(JobID jobId)	No. An exception is reported.
JobClient	getStagingAreaDir()	No. An exception is reported.
JobClient	getSystemDir()	No. An exception is reported.
JobClient	getTaskOutputFilter()	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce
JobClient	getTaskOutputFilter(JobConf job)	No. An exception is reported.
JobClient	init(JobConf conf)	No. An exception is reported.
JobClient	isJobDirValid(Path jobDirPath, FileSystem fs)	No. An exception is reported.
JobClient	jobsToComplete()	No. An exception is reported.
JobClient	monitorAndPrintJob(JobConf conf, RunningJob job)	No. An exception is reported.
JobClient	renewDelegationToken(Token<DelegationTokenIdentifier> token)	No. An exception is reported.
JobClient	run(String[] argv)	No. An exception is reported.
JobClient	runJob(JobConf job)	Yes.
JobClient	setTaskOutputFilter(JobClient.TaskStatusFilter newValue)	No. An exception is reported.
JobClient	setTaskOutputFilter(JobConf job, JobClient.TaskStatusFilter newValue)	No. An exception is reported.
JobClient	submitJob(JobConf job)	Yes.
JobClient	submitJob(String jobFile)	No. An exception is reported.
JobConf	deleteLocalFiles()	No. An exception is reported.
JobConf	deleteLocalFiles(String subdir)	No. An exception is reported.
JobConf	normalizeMemoryConfigValue(long val)	Empty implementation.
JobConf	setCombinerClass(Class<? extends Reducer> theClass)	Yes.
JobConf	setCompressMapOutput(boolean compress)	Empty implementation.
JobConf	setInputFormat(Class<? extends InputFormat> theClass)	No. An exception is reported.
JobConf	setJar(String jar)	No. An exception is reported.
JobConf	setJarByClass(Class cls)	No. An exception is reported.
JobConf	setJobEndNotificationURI(String uri)	No. An exception is reported.
JobConf	setJobName(String name)	Empty implementation.
JobConf	setJobPriority(JobPriority prio)	No. An exception is reported.
JobConf	setKeepFailedTaskFiles(boolean keep)	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce
JobConf	setKeepTaskFilesPattern(String pattern)	No. An exception is reported.
JobConf	setKeyFieldComparatorOptions(String keySpec)	No. An exception is reported.
JobConf	setKeyFieldPartitionerOptions(String keySpec)	No. An exception is reported.
JobConf	setMapDebugScript(String mDbgScript)	Empty implementation.
JobConf	setMapOutputCompressorClass(Class<? extends CompressionCodec> codecClass)	Empty implementation.
JobConf	setMapOutputKeyClass(Class<? > theClass)	Yes.
JobConf	setMapOutputValueClass(Class<? > theClass)	Yes.
JobConf	setMapperClass(Class<? extends Mapper> theClass)	Yes.
JobConf	setMapRunnerClass(Class<? extends MapRunnable> theClass)	No. An exception is reported.
JobConf	setMapSpeculativeExecution(boolean speculativeExecution)	Empty implementation.
JobConf	setMaxMapAttempts(int n)	Empty implementation.
JobConf	setMaxMapTaskFailuresPercent(int percent)	Empty implementation.
JobConf	setMaxPhysicalMemoryForTask(long mem)	Empty implementation.
JobConf	setMaxReduceAttempts(int n)	Empty implementation.
JobConf	setMaxReduceTaskFailuresPercent(int percent)	Empty implementation.
JobConf	setMaxTaskFailuresPerTracker(int noFailures)	Empty implementation.
JobConf	setMaxVirtualMemoryForTask(long vmem)	Empty implementation.
JobConf	setMemoryForMapTask(long mem)	Yes.
JobConf	setMemoryForReduceTask(long mem)	Yes.
JobConf	setNumMapTasks(int n)	Yes.

Type	Interface	Compatible with Hadoop MapReduce
JobConf	setNumReduceTasks(int n)	Yes.
JobConf	setNumTasksToExecutePerJvm(int numTasks)	Empty implementation.
JobConf	setOutputCommitter(Class<? extends OutputCommitter> theClass)	No. An exception is reported.
JobConf	setOutputFormat(Class<? extends OutputFormat> theClass)	Empty implementation.
JobConf	setOutputKeyClass(Class<? > theClass)	Yes.
JobConf	setOutputKeyComparatorClass(Class<? extends RawComparator> theClass)	No. An exception is reported.
JobConf	setOutputValueClass(Class<? > theClass)	Yes.
JobConf	setOutputValueGroupingComparator(Class<? extends RawComparator> theClass)	No. An exception is reported.
JobConf	setPartitionerClass(Class<? extends Partitioner> theClass)	Yes.
JobConf	setProfileEnabled(boolean newValue)	Empty implementation.
JobConf	setProfileParams(String value)	Empty implementation.
JobConf	setProfileTaskRange(boolean isMap, String newValue)	Empty implementation.
JobConf	setQueueName(String queueName)	No. An exception is reported.
JobConf	setReduceDebugScript(String rDbgScript)	Empty implementation.
JobConf	setReducerClass(Class<? extends Reducer> theClass)	Yes.
JobConf	setReduceSpeculativeExecution(boolean speculativeExecution)	Empty implementation.
JobConf	setSessionId(String sessionId)	Empty implementation.
JobConf	setSpeculativeExecution(boolean speculativeExecution)	No. An exception is reported.
JobConf	setUseNewMapper(boolean flag)	Yes.
JobConf	setUseNewReducer(boolean flag)	Yes.

Type	Interface	Compatible with Hadoop MapReduce
JobConf	setUser(String user)	Empty implementation.
JobConf	setWorkingDirectory(Path dir)	Empty implementation.
FileInputFormat	N/A	No. An exception is reported.
TextInputFormat	N/A	Yes.
InputSplit	mapred.min.split.size.	No. An exception is reported.
FileSplit	map.input.file	No. An exception is reported.
RecordWriter	N/A	No. An exception is reported.
RecordReader	N/A	No. An exception is reported.
OutputFormat	N/A	No. An exception is reported.
OutputCommitter	abortJob(JobContext jobContext, int status)	No. An exception is reported.
OutputCommitter	abortJob(JobContext context, JobStatus.State runState)	No. An exception is reported.
OutputCommitter	abortTask(TaskAttemptContext taskContext)	No. An exception is reported.
OutputCommitter	abortTask(TaskAttemptContext taskContext)	No. An exception is reported.
OutputCommitter	cleanupJob(JobContext jobContext)	No. An exception is reported.
OutputCommitter	cleanupJob(JobContext context)	No. An exception is reported.
OutputCommitter	commitJob(JobContext jobContext)	No. An exception is reported.
OutputCommitter	commitJob(JobContext context)	No. An exception is reported.
OutputCommitter	commitTask(TaskAttemptContext taskContext)	No. An exception is reported.
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	No. An exception is reported.
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	No. An exception is reported.
OutputCommitter	setupJob(JobContext jobContext)	No. An exception is reported.
OutputCommitter	setupJob(JobContext jobContext)	No. An exception is reported.
OutputCommitter	setupTask(TaskAttemptContext taskContext)	No. An exception is reported.
OutputCommitter	setupTask(TaskAttemptContext taskContext)	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce
Counter	getDisplayName()	Yes.
Counter	getName()	Yes.
Counter	getValue()	Yes.
Counter	increment(long incr)	Yes.
Counter	setValue(long value)	Yes.
Counter	setDisplayDisplayName(String displayName)	Yes.
DistributedCache	CACHE_ARCHIVES	No. An exception is reported.
DistributedCache	CACHE_ARCHIVES_SIZES	No. An exception is reported.
DistributedCache	CACHE_ARCHIVES_TIMESTAMPS	No. An exception is reported.
DistributedCache	CACHE_FILES	No. An exception is reported.
DistributedCache	CACHE_FILES_SIZES	No. An exception is reported.
DistributedCache	CACHE_FILES_TIMESTAMPS	No. An exception is reported.
DistributedCache	CACHE_LOCALARCHIVES	No. An exception is reported.
DistributedCache	CACHE_LOCALFILES	No. An exception is reported.
DistributedCache	CACHE_SYMLINK	No. An exception is reported.
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf)	No. An exception is reported.
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf, FileSystem fs)	No. An exception is reported.
DistributedCache	addCacheArchive(URI uri, Configuration conf)	No. An exception is reported.
DistributedCache	addCacheFile(URI uri, Configuration conf)	No. An exception is reported.
DistributedCache	addFileToClassPath(Path file, Configuration conf)	No. An exception is reported.
DistributedCache	addFileToClassPath(Path file, Configuration conf, FileSystem fs)	No. An exception is reported.
DistributedCache	addLocalArchives(Configuration conf, String str)	No. An exception is reported.
DistributedCache	addLocalFiles(Configuration conf, String str)	No. An exception is reported.
DistributedCache	checkURIs(URI[] uriFiles, URI[] uriArchives)	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce
DistributedCache	createAllSymlink(Configuration conf, File jobCacheDir, File workDir)	No. An exception is reported.
DistributedCache	createSymlink(Configuration conf)	No. An exception is reported.
DistributedCache	getArchiveClassPaths(Configuration conf)	No. An exception is reported.
DistributedCache	getArchiveTimestamps(Configuration conf)	No. An exception is reported.
DistributedCache	getCacheArchives(Configuration conf)	No. An exception is reported.
DistributedCache	getCacheFiles(Configuration conf)	No. An exception is reported.
DistributedCache	getFileClassPaths(Configuration conf)	No. An exception is reported.
DistributedCache	getFileStatus(Configuration conf, URI cache)	No. An exception is reported.
DistributedCache	getFileTimestamps(Configuration conf)	No. An exception is reported.
DistributedCache	getLocalCacheArchives(Configuration conf)	No. An exception is reported.
DistributedCache	getLocalCacheFiles(Configuration conf)	No. An exception is reported.
DistributedCache	getSymlink(Configuration conf)	No. An exception is reported.
DistributedCache	getTimestamp(Configuration conf, URI cache)	No. An exception is reported.
DistributedCache	setArchiveTimestamps(Configuration conf, String timestamps)	No. An exception is reported.
DistributedCache	setCacheArchives(URI[] archives, Configuration conf)	No. An exception is reported.
DistributedCache	setCacheFiles(URI[] files, Configuration conf)	No. An exception is reported.
DistributedCache	setFileTimestamps(Configuration conf, String timestamps)	No. An exception is reported.
DistributedCache	setLocalArchives(Configuration conf, String str)	No. An exception is reported.
DistributedCache	setLocalFiles(Configuration conf, String str)	No. An exception is reported.
IsolationRunner	N/A	No. An exception is reported.
Profiling	N/A	Empty implementation.

Type	Interface	Compatible with Hadoop MapReduce
Debugging	N/A	Empty implementation.
Data Compression	N/A	Yes.
Skipping Bad Records	N/A	No. An exception is reported.
Job Authorization	mapred.acls.enabled	No. An exception is reported.
Job Authorization	mapreduce.job.acl-view-job	No. An exception is reported.
Job Authorization	mapreduce.job.acl-modify-job	No. An exception is reported.
Job Authorization	mapreduce.cluster.administrators	No. An exception is reported.
Job Authorization	mapred.queue.queue-name.acl-administer-jobs	No. An exception is reported.
MultipleInputs	N/A	No. An exception is reported.
MultipleOutputs	N/A	Yes.
org.apache.hadoop.mapreduce.lib.db	N/A	No. An exception is reported.
org.apache.hadoop.mapreduce.security	N/A	No. An exception is reported.
org.apache.hadoop.mapreduce.lib.jobcontrol	N/A	No. An exception is reported.
org.apache.hadoop.mapreduce.lib.chain	N/A	No. An exception is reported.
org.apache.hadoop.mapreduce.lib.db	N/A	No. An exception is reported.

8.5. Data types

This topic describes the data types supported by MapReduce.

MapReduce supports the following data types: BIGINT, DOUBLE, STRING, DATETIME, BOOLEAN, and DECIMAL. The following table lists the mappings between MaxCompute data types and Java data types.

MaxCompute SQL Type	Java Type
BIGINT	LONG
DOUBLE	DOUBLE
DECIMAL	BIGDECIMAL
BOOLEAN	BOOLEAN
STRING	STRING
DATETIME	DATE

8.6. Sample programs

8.6.1. WordCount example

This topic provides a WordCount example.

Example:

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
public class WordCount {
    public static class TokenizerMapper extends MapperBase {
        private Record word;
        private Record one;
        @Override
        public void setup(TaskContext context) throws IOException {
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
            System.out.println("TaskID:" + context.getTaskID().toString());
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                word.set(new Object[] { record.get(i).toString() });
                context.write(word, one);
            }
        }
    }
    /**
     * A combiner class that combines map output by sum them.
     */
    public static class SumCombiner extends ReducerBase {
        private Record count;
        @Override
        public void setup(TaskContext context) throws IOException {
            count = context.createMapOutputValueRecord();
        }
        // The combiner implements the same interface as that of the reducer. The combiner allows you
        // to immediately run a local reduce job on the mapper to reduce the output of the mapper.
        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long c = 0;
            while (values.hasNext()) {
                Record val = values.next();
                c += (Long) val.get(0);
            }
        }
    }
}
```

```

    }
    count.set(0, c);
    context.write(key, count);
  }
}
/**
 * A reducer class that just emits the sum of the input values.
 **/
public static class SumReducer extends ReducerBase {
  private Record result = null;
  @Override
  public void setup(TaskContext context) throws IOException {
    result = context.createOutputRecord();
  }
  @Override
  public void reduce(Record key, Iterator<Record> values, TaskContext context)
    throws IOException {
    long count = 0;
    while (values.hasNext()) {
      Record val = values.next();
      count += (Long) val.get(0);
    }
    result.set(0, key.get(0));
    result.set(1, count);
    context.write(result);
  }
}
public static void main(String[] args) throws Exception {
  if (args.length != 2) {
    System.err.println("Usage: WordCount <in_table> <out_table>");
    System.exit(2);
  }
  JobConf job = new JobConf();
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(SumCombiner.class);
  job.setReducerClass(SumReducer.class);
  // Configure the schema that defines the intermediate output of the mapper as key-value pairs.
  // The intermediate output of the mapper exists as records.
  job.setMapOutputKeySchema (SchemaUtils.fromString("word:string"));
  job.setMapOutputValueSchema (SchemaUtils.fromString("count:bigint"));
  // Configure information about input and output tables.
  InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
  OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
  JobClient.runJob(job);
}
}

```

8.6.2. MapOnly example

This topic provides a MapOnly example of MapReduce.

For MapOnly jobs, a mapper directly generates key-value pairs to MaxCompute tables. You need only to specify output tables. You do not need to specify the key-value metadata for the output of a mapper.

Example:

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import com.aliyun.odps.data.Record;

```

```

import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
public class MapOnly {
    public static class MapperClass extends MapperBase {
        @Override
        public void setup(TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.setup", false);
            // The main function executes the following logic only if option.mapper.setup is set to
            true in the JobConf file:
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, "setup");
                result.set(1, 1L);
                context.write(result);
            }
        }
        @Override
        public void map(long key, Record record, TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.map", false);
            // The main function executes the following logic only if option.mapper.map is set to tr
            ue in the JobConf file:
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, record.get(0));
                result.set(1, 1L);
                context.write(result);
            }
        }
        @Override
        public void cleanup(TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.cleanup", false);
            // The main function executes the following logic only if option.mapper.cleanup is set t
            o true in the JobConf file:
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, "cleanup");
                result.set(1, 1L);
                context.write(result);
            }
        }
    }
}
public static void main(String[] args) throws Exception {
    if (args.length != 2 && args.length != 3) {
        System.err.println("Usage: OnlyMapper <in_table> <out_table> [setup|map|cleanup]");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(MapperClass.class);
    // For MapOnly jobs, the number of reducers must be explicitly set to 0.
    job.setNumReduceTasks(0);
    // Configure information about input and output tables.
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    if (args.length == 3) {

```

```

    String options = new String(args[2]);
    // You can specify key-value pairs in the JobConf file, and use getJobConf of the context to query the configurations in a mapper.
    if (options.contains("setup")) {
        job.setBoolean("option.mapper.setup", true);
    }
    if (options.contains("map")) {
        job.setBoolean("option.mapper.map", true);
    }
    if (options.contains("cleanup")) {
        job.setBoolean("option.mapper.cleanup", true);
    }
}
JobClient.runJob(job);
}
}

```

8.6.3. MultipleInOut example

This topic provides a MultipleInOut example.

MaxCompute jobs can read data from multiple input tables and write data to multiple output tables. To read data from multiple input tables, you must make sure that the input tables have the same number of columns and the same data types. These requirements do not apply to output tables.

Example:

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedHashMap;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * Multi input & output example.
 */
public class MultipleInOut {
    public static class TokenizerMapper extends MapperBase {
        Record word;
        Record one;
        @Override
        public void setup(TaskContext context) throws IOException {
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                word.set(i, record.get(i).toString());
            }
        }
    }
}

```

```

        word.set(new Object[] { record.get(1).toString() });
        context.write(word, one);
    }
}
}
public static class SumReducer extends ReducerBase {
    private Record result;
    private Record result1;
    private Record result2;
    @Override
    public void setup(TaskContext context) throws IOException {
        // Create a record for each output and add labels to distinguish outputs.
        result = context.createOutputRecord();
        result1 = context.createOutputRecord("out1");
        result2 = context.createOutputRecord("out2");
    }
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context)
        throws IOException {
        long count = 0;
        while (values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        long mod = count % 3;
        if (mod == 0) {
            result.set(0, key.get(0));
            result.set(1, count);
            // If you do not specify a label, the default output is used.
            context.write(result);
        } else if (mod == 1) {
            result1.set(0, key.get(0));
            result1.set(1, count);
            context.write(result1, "out1");
        } else {
            result2.set(0, key.get(0));
            result2.set(1, count);
            context.write(result2, "out2");
        }
    }
    @Override
    public void cleanup(TaskContext context) throws IOException {
        Record result = context.createOutputRecord();
        result.set(0, "default");
        result.set(1, 1L);
        context.write(result);
        Record result1 = context.createOutputRecord("out1");
        result1.set(0, "out1");
        result1.set(1, 1L);
        context.write(result1, "out1");
        Record result2 = context.createOutputRecord("out2");
        result2.set(0, "out2");
        result2.set(1, 1L);
        context.write(result2, "out2");
    }
}
// Convert partition strings such as "ds=1/pt=2" to MAP.
public static LinkedHashMap<String, String> convertPartSpecToMap(
    String partSpec) {
    LinkedHashMap<String, String> map = new LinkedHashMap<String, String>();
}

```

```

LinkedHashMap<String, String> map = new LinkedHashMap<String, String>();
if (partSpec != null && ! partSpec.trim().isEmpty()) {
    String[] parts = partSpec.split("/");
    for (String part : parts) {
        String[] ss = part.split("=");
        if (ss.length != 2) {
            throw new RuntimeException("ODPS-0730001: error part spec format: "
                + partSpec);
        }
        map.put(ss[0], ss[1]);
    }
}
return map;
}

public static void main(String[] args) throws Exception {
    String[] inputs = null;
    String[] outputs = null;
    if (args.length == 2) {
        inputs = args[0].split(",");
        outputs = args[1].split(",");
    } else {
        System.err.println("MultipleInOut in... out...");
        System.exit(1);
    }
    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    // Parse input table strings.
    for (String in : inputs) {
        String[] ss = in.split("\\|");
        if (ss.length == 1) {
            InputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
        } else if (ss.length == 2) {
            LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
            InputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
        } else {
            System.err.println("Style of input: " + in + " is not right");
            System.exit(1);
        }
    }
    // Parse output table strings.
    for (String out : outputs) {
        String[] ss = out.split("\\|");
        if (ss.length == 1) {
            OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
        } else if (ss.length == 2) {
            LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
            OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
        } else if (ss.length == 3) {
            if (ss[1].isEmpty()) {
                LinkedHashMap<String, String> map = convertPartSpecToMap(ss[2]);
                OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(),
                    job);
            } else {
                LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
                OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(),
                    job);
            }
        }
    }
}

```



```

        v = (Long) i.get(v);
        if (expect != v) {
            throw new IOException("expect: " + expect + ", but: " + v);
        }
        count++;
    }
    if (count != 1) {
        throw new IOException("res_table should have 1 record, but: " + count);
    }
    Record record = context.createOutputRecord();
    v--;
    record.set(0, v);
    context.write(record);
    // Set the counter. The counter value can be obtained in the main function after the job
    is complete.
    context.getCounter("multijobs", "value").setValue(v);
}
}
public static void main(String[] args) throws Exception {
    if (args.length != 1) {
        System.err.println("Usage: TestMultiJobs <table>");
        System.exit(1);
    }
    String tbl = args[0];
    long iterCount = 2;
    System.err.println("Start to run init job.");
    JobConf initJob = new JobConf();
    initJob.setLong("multijobs.value", iterCount);
    initJob.setMapperClass(InitMapper.class);
    InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), initJob);
    OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), initJob);
    initJob.setMapOutputKeySchema(SchemaUtils.fromString("key:string"));
    initJob.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
    // Explicitly set the number of reducers to 0 for map-only jobs.
    initJob.setNumReduceTasks(0);
    JobClient.runJob(initJob);
    while (true) {
        System.err.println("Start to run iter job, count: " + iterCount);
        JobConf decJob = new JobConf();
        decJob.setLong("multijobs.expect.value", iterCount);
        decJob.setMapperClass(DecreaseMapper.class);
        InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), decJob);
        OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), decJob);
        // Explicitly set the number of reducers to 0 for map-only jobs.
        decJob.setNumReduceTasks(0);
        RunningJob rJob = JobClient.runJob(decJob);
        iterCount--;
        // If the specified number of iterations is reached, exit the loop.
        if (rJob.getCounters().findCounter("multijobs", "value").getValue() == 0) {
            break;
        }
    }
    if (iterCount != 0) {
        throw new IOException("Job failed.");
    }
}
}
}

```

8.6.5. SecondarySort example

This topic provides a SecondarySort example.

Example:

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
/**
 *
 * This is an example ODPS Map/Reduce application. It reads the input table that
 * must contain two integers per record. The output is sorted by the first and
 * second number and grouped on the first number.
 *
 */
public class SecondarySort {
    /**
     * Read two integers from each line and generate a key, value pair as ((left,
     * right), right).
     */
    public static class MapClass extends MapperBase {
        private Record key;
        private Record value;
        @Override
        public void setup(TaskContext context) throws IOException {
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            long left = 0;
            long right = 0;
            if (record.getColumnCount() > 0) {
                left = (Long) record.get(0);
                if (record.getColumnCount() > 1) {
                    right = (Long) record.get(1);
                }
                key.set(new Object[] { (Long) left, (Long) right });
                value.set(new Object[] { (Long) right });
                context.write(key, value);
            }
        }
    }
    /**
     * A reducer class that just emits the sum of the input values.
     */
    public static class ReduceClass extends ReducerBase {
```

```
private Record result = null;
@Override
public void setup(TaskContext context) throws IOException {
    result = context.createOutputRecord();
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context)
    throws IOException {
    result.set(0, key.get(0));
    while (values.hasNext()) {
        Record value = values.next();
        result.set(1, value.get(0));
        context.write(result);
    }
}
}

public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: secondarysrot <in> <out>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(MapClass.class);
    job.setReducerClass(ReduceClass.class);
    // Set multiple columns as keys.
    //compare first and second parts of the pair
    job.setOutputKeySortColumns(new String[] { "i1", "i2" });
    //partition based on the first part of the pair
    job.setPartitionColumns(new String[] { "i1" });
    //grouping comparator based on the first part of the pair
    job.setOutputGroupingColumns(new String[] { "i1" });
    //the map output is LongPair, Long
    job.setMapOutputKeySchema (SchemaUtils.fromString("i1:bigint,i2:bigint"));
    job.setMapOutputValueSchema (SchemaUtils.fromString("i2x:bigint"));
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    JobClient.runJob(job);
    System.exit(0);
}
}
```

8.6.6. Resource usage example

This topic provides an example of using resources in MapReduce.

Example:

```
package com.aliyun.odps.mapred.open.example;
import java.io.BufferedInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
```

```
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * Upload
 *
 * Import data from text file into table
 *
 */
public class Upload {
    public static class UploadMapper extends MapperBase {
        @Override
        public void setup(TaskContext context) throws IOException {
            Record record = context.createOutputRecord();
            StringBuilder importdata = new StringBuilder();
            BufferedInputStream bufferedInput = null;
            try {
                byte[] buffer = new byte[1024];
                int bytesRead = 0;
                String filename = context.getJobConf().get("import.filename");
                bufferedInput = context.readResourceFileAsStream(filename);
                while ((bytesRead = bufferedInput.read(buffer)) != -1) {
                    String chunk = new String(buffer, 0, bytesRead);
                    importdata.append(chunk);
                }
                String lines[] = importdata.toString().split("\n");
                for (int i = 0; i < lines.length; i++) {
                    String[] ss = lines[i].split(",");
                    record.set(0, Long.parseLong(ss[0].trim()));
                    record.set(1, ss[1].trim());
                    context.write(record);
                }
            } catch (FileNotFoundException ex) {
                throw new IOException(ex);
            } catch (IOException ex) {
                throw new IOException(ex);
            } finally {
            }
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: Upload <import_txt> <out_table>");
            System.exit(2);
        }
        JobConf job = new JobConf();
        job.setMapperClass(UploadMapper.class);
        // Specify the resource name, which can be obtained in the map stage by using the JobConf in
        interface.
        job.set("import.filename", args[0]);
        // Explicitly set the number of reducers to 0 for MapOnly jobs.
        job.setNumReduceTasks(0);
        job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
        InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    }
}
```

```
        JobClient.runJob(job);
    }
}
```

Note

You can use one of the following methods to set the JobConf file:

- Use the JobConf interface in the SDK. This method is used in this example.
- Use the `-conf` parameter in a jar command to specify a new JobConf file.

8.6.7. Counter usage example

This topic provides an example of using counters in MapReduce.

Three counters are defined in this example: `map_outputs`, `reduce_outputs`, and `global_counts`. You can use the `setup`, `map`, `reduce`, and `cleanup` APIs of the Map or Reduce function to obtain custom counters and perform related operations.

Example:

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.counter.Counters;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
/**
 *
 * User Defined Counters
 *
 */
public class UserDefinedCounters {
    enum MyCounter {
        TOTAL_TASKS, MAP_TASKS, REDUCE_TASKS
    }
    public static class TokenizerMapper extends MapperBase {
        private Record word;
        private Record one;
        @Override
        public void setup(TaskContext context) throws IOException {
            super.setup(context);
            Counter map_tasks = context.getCounter(MyCounter.MAP_TASKS);
            Counter total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
            map_tasks.increment(1);
            total_tasks.increment(1);
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L, 1L });
        }
    }
}
```

```

        one.set(new Object[] { 1 });
    }
    @Override
    public void map(long recordNum, Record record, TaskContext context)
        throws IOException {
        for (int i = 0; i < record.getColumnCount(); i++) {
            word.set(new Object[] { record.get(i).toString() });
            context.write(word, one);
        }
    }
}

public static class SumReducer extends ReducerBase {
    private Record result = null;
    @Override
    public void setup(TaskContext context) throws IOException {
        result = context.createOutputRecord();
        Counter reduce_tasks = context.getCounter(MyCounter.REDUCE_TASKS);
        Counter total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
        reduce_tasks.increment(1);
        total_tasks.increment(1);
    }
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context)
        throws IOException {
        long count = 0;
        while (values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        result.set(0, key.get(0));
        result.set(1, count);
        context.write(result);
    }
}

public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err
            .println("Usage: TestUserDefinedCounters <in_table> <out_table>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeySchema (SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema (SchemaUtils.fromString("count:bigint"));
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    RunningJob rJob = JobClient.runJob(job);
    // If the job is successful, the values of the custom counters in the job are returned.
    Counters counters = rJob.getCounters();
    long m = counters.findCounter(MyCounter.MAP_TASKS).getValue();
    long r = counters.findCounter(MyCounter.REDUCE_TASKS).getValue();
    long total = counters.findCounter(MyCounter.TOTAL_TASKS).getValue();
    System.exit(0);
}
}

```

8.6.8. Grep example

This topic provides a Grep example.

Example:

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.Mapper;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 *
 * Extracts matching regexs from input files and counts them.
 *
 */
public class Grep {
    /**
     * RegexMapper
     */
    public class RegexMapper extends MapperBase {
        private Pattern pattern;
        private int group;
        private Record word;
        private Record one;
        @Override
        public void setup(TaskContext context) throws IOException {
            JobConf job = (JobConf) context.getJobConf();
            pattern = Pattern.compile(job.get("mapred.mapper.regex"));
            group = job.getInt("mapred.mapper.regex.group", 0);
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context) throws IOException {
            for (int i = 0; i < record.getColumnCount(); ++i) {
                String text = record.get(i).toString();
                Matcher matcher = pattern.matcher(text);
                while (matcher.find()) {
                    word.set(new Object[] { matcher.group(group) });
                    context.write(word, one);
                }
            }
        }
    }
}
/**
```

```

    * LongSumReducer
    **/
public class LongSumReducer extends ReducerBase {
    private Record result = null;
    @Override
    public void setup(TaskContext context) throws IOException {
        result = context.createOutputRecord();
    }
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOEx
ception {
        long count = 0;
        while (values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        result.set(0, key.get(0));
        result.set(1, count);
        context.write(result);
    }
}
/**
 * A {@link Mapper} that swaps keys and values.
 **/
public class InverseMapper extends MapperBase {
    private Record word;
    private Record count;
    @Override
    public void setup(TaskContext context) throws IOException {
        word = context.createMapOutputValueRecord();
        count = context.createMapOutputKeyRecord();
    }
    /**
     * The inverse function. Input keys and values are swapped.
     **/
    @Override
    public void map(long recordNum, Record record, TaskContext context) throws IOException {
        word.set(new Object[] { record.get(0).toString() });
        count.set(new Object[] { (Long) record.get(1) });
        context.write(count, word);
    }
}
/**
 * IdentityReducer
 **/
public class IdentityReducer extends ReducerBase {
    private Record result = null;
    @Override
    public void setup(TaskContext context) throws IOException {
        result = context.createOutputRecord();
    }
    /** Writes all keys and values directly to output. **/
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOEx
ception {
        result.set(0, key.get(0));
        while (values.hasNext()) {
            Record val = values.next();
            result.set(1, val.get(0));
        }
    }
}

```

```

        context.write(result);
    }
}
}
public static void main(String[] args) throws Exception {
    if (args.length < 4) {
        System.err.println("Grep <inDir> <tmpDir> <outDir> <regex> [<group>]");
        System.exit(2);
    }
    JobConf grepJob = new JobConf();
    grepJob.setMapperClass(RegexMapper.class);
    grepJob.setReducerClass(LongSumReducer.class);
    grepJob.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    grepJob.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), grepJob);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), grepJob);
    // Set the regular expression for grepJob.
    grepJob.set("mapred.mapper.regex", args[3]);
    if (args.length == 5) {
        grepJob.set("mapred.mapper.regex.group", args[4]);
    }
    @SuppressWarnings("unused")
    RunningJob rjGrep = JobClient.runJob(grepJob);
    // Specify the output of grepJob as the input of sortJob.
    JobConf sortJob = new JobConf();
    sortJob.setMapperClass(InverseMapper.class);
    sortJob.setReducerClass(IdentityReducer.class);
    sortJob.setMapOutputKeySchema(SchemaUtils.fromString("count:bigint"));
    sortJob.setMapOutputValueSchema(SchemaUtils.fromString("word:string"));
    InputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), sortJob);
    OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), sortJob);
    sortJob.setNumReduceTasks(1); // write a single file
    sortJob.setOutputKeySortColumns(new String[] { "count" });
    @SuppressWarnings("unused")
    RunningJob rjSort = JobClient.runJob(sortJob);
}
}
}

```

8.6.9. Join example

This topic provides a Join example.

The MaxCompute MapReduce framework does not support Join operations. However, you can join data by using a custom map or reduce function.

Example:

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;

```

```
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * Join, mr_Join_src1/mr_Join_src2(key bigint, value string), mr_Join_out(key
 * bigint, value1 string, value2 string)
 *
 */
public class Join {
    public static final Log LOG = LogFactory.getLog(Join.class);
    public static class JoinMapper extends MapperBase {
        private Record mapkey;
        private Record mapvalue;
        private long tag;
        @Override
        public void setup(TaskContext context) throws IOException {
            mapkey = context.createMapOutputKeyRecord();
            mapvalue = context.createMapOutputValueRecord();
            tag = context.getInputTableInfo().getLabel().equals("left") ? 0 : 1;
        }
        @Override
        public void map(long key, Record record, TaskContext context)
            throws IOException {
            mapkey.set(0, record.get(0));
            mapkey.set(1, tag);
            for (int i = 1; i < record.getColumnCount(); i++) {
                mapvalue.set(i - 1, record.get(i));
            }
            context.write(mapkey, mapvalue);
        }
    }
    public static class JoinReducer extends ReducerBase {
        private Record result = null;
        @Override
        public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        // Each input of the reduce function is the records that have the same key.
        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long k = key.getBigint(0);
            List<Object[]> leftValues = new ArrayList<Object[]>();
            // Records are sorted based on the combination of the key and tag. This ensures that rec
            ords in the left table are passed to the reduce function first when the reduce function performs the
            Join operation.
            while (values.hasNext()) {
                Record value = values.next();
                long tag = (Long) key.get(1);
                // Data in the left table is first cached in memory.
                if (tag == 0) {
                    leftValues.add(value.toArray().clone());
                } else {
                    // Data in the right table is joined with all data in the left table.
                    // The sample code has poor performance and is only used as an example. We recom
                    mend that you do not use the code in your production environment.
                    for (Object[] leftValue : leftValues) {
```



```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.conf.JobConf;
public class Sleep {
    private static final String SLEEP_SECS = "sleep.secs";
    public static class MapperClass extends MapperBase {
        // No data is entered, the map function is not executed, and the related logic can be written on
        // ly into setup.
        @Override
        public void setup(TaskContext context) throws IOException {
            try {
                // Obtain the number of sleep seconds set in JobConf.
                Thread.sleep(context.getJobConf().getInt(SLEEP_SECS, 1) * 1000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: Sleep <sleep_secs>");
            System.exit(-1);
        }
        JobConf job = new JobConf();
        job.setMapperClass(MapperClass.class);
        // This instance is also a MapOnly job and the number of reducers must be set to 0.
        job.setNumReduceTasks(0);
        // The number of mappers must be specified by the user because no input table is provided.
        job.setNumMapTasks(1);
        job.set(SLEEP_SECS, args[0]);
        JobClient.runJob(job);
    }
}
```

8.6.11. Unique example

This topic provides a Unique example.

Example:

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * Unique Remove duplicate words
 */
```

```

    **/
public class Unique {
    public static class OutputSchemaMapper extends MapperBase {
        private Record key;
        private Record value;
        @Override
        public void setup(TaskContext context) throws IOException {
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            long left = 0;
            long right = 0;
            if (record.getColumnCount() > 0) {
                left = (Long) record.get(0);
                if (record.getColumnCount() > 1) {
                    right = (Long) record.get(1);
                }
                key.set(new Object[] { (Long) left, (Long) right });
                value.set(new Object[] { (Long) left, (Long) right });
                context.write(key, value);
            }
        }
    }
    public static class OutputSchemaReducer extends ReducerBase {
        private Record result = null;
        @Override
        public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            result.set(0, key.get(0));
            while (values.hasNext()) {
                Record value = values.next();
                result.set(1, value.get(1));
            }
            context.write(result);
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length > 3 || args.length < 2) {
            System.err.println("Usage: unique <in> <out> [key|value|all]");
            System.exit(2);
        }
        String ops = "all";
        if (args.length == 3) {
            ops = args[2];
        }
        // The input group of Reduce is determined by the value of the setOutputGroupingColumns parameter. If this parameter is not specified, the default value MapOutputKeySchema is used.
        // Key Unique
        if (ops.equals("key")) {
            JobConf job = new JobConf();
            job.setMapperClass(OutputSchemaMapper.class);
            job.setReducerClass(OutputSchemaReducer.class);

```

```

        job.setMapOutputKeySchema (SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setMapOutputValueSchema (SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setPartitionColumns(new String[] { "key" });
        job.setOutputKeySortColumns(new String[] { "key", "value" });
        job.setOutputGroupingColumns(new String[] { "key" });
        job.set("tablename2", args[1]);
        job.setNumReduceTasks (1);
        job.setInt("table.counter", 0);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        JobClient.runJob(job);
    }
    // Key&Value Unique
    if (ops.equals("all")) {
        JobConf job = new JobConf();
        job.setMapperClass (OutputSchemaMapper.class);
        job.setReducerClass (OutputSchemaReducer.class);
        job.setMapOutputKeySchema (SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setMapOutputValueSchema (SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setPartitionColumns(new String[] { "key" });
        job.setOutputKeySortColumns(new String[] { "key", "value" });
        job.setOutputGroupingColumns(new String[] { "key", "value" });
        job.set("tablename2", args[1]);
        job.setNumReduceTasks (1);
        job.setInt("table.counter", 0);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        JobClient.runJob(job);
    }
    // Value Unique
    if (ops.equals("value")) {
        JobConf job = new JobConf();
        job.setMapperClass (OutputSchemaMapper.class);
        job.setReducerClass (OutputSchemaReducer.class);
        job.setMapOutputKeySchema (SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setMapOutputValueSchema (SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setPartitionColumns(new String[] { "value" });
        job.setOutputKeySortColumns(new String[] { "value" });
        job.setOutputGroupingColumns(new String[] { "value" });
        job.set("tablename2", args[1]);
        job.setNumReduceTasks (1);
        job.setInt("table.counter", 0);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        JobClient.runJob(job);
    }
}
}
}

```

8.6.12. Sort example

This topic provides a Sort example.

Example:

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Date;

```

```
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.example.lib.IdentityReducer;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * This is the trivial map/reduce program that does absolutely nothing other
 * than use the framework to fragment and sort the input values.
 *
 **/
public class Sort {
    static int printUsage() {
        System.out.println("sort <input> <output>");
        return -1;
    }
    /**
     * Implements the identity function, mapping record's first two columns to
     * outputs.
     **/
    public static class IdentityMapper extends MapperBase {
        private Record key;
        private Record value;
        @Override
        public void setup(TaskContext context) throws IOException {
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            key.set(new Object[] { (Long) record.get(0) });
            value.set(new Object[] { (Long) record.get(1) });
            context.write(key, value);
        }
    }
    /**
     * The main driver for sort program. Invoke this method to submit the
     * map/reduce job.
     *
     * @throws IOException
     *         When there is communication problems with the job tracker.
     **/
    public static void main(String[] args) throws Exception {
        JobConf jobConf = new JobConf();
        jobConf.setMapperClass(IdentityMapper.class);
        jobConf.setReducerClass(IdentityReducer.class);
        // For global sorting, the number of reducers is set to 1. All the data is transferred to the same
        reducer.
        // This method applies only to the scenarios when small amounts of data are processed. If large am
        ounts of data need to be processed, use other methods, such as TeraSort.
        jobConf.setNumReduceTasks(1);
        jobConf.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint"));
        jobConf.setMapOutputValueSchema(SchemaUtils.fromString("value:bigint"));
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), jobConf);
    }
}
```

```
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), jobConf);
Date startTime = new Date();
System.out.println("Job started: " + startTime);
JobClient.runJob(jobConf);
Date endTime = new Date();
System.out.println("Job ended: " + endTime);
System.out.println("The job took " + (endTime.getTime() - startTime.getTime()) / 1000 + " seconds
.");
}
}
```

8.6.13. Examples of using partitioned tables as input

This topic provides examples of using partitioned tables as input.

Example 1:

```
public static void main(String[] args) throws Exception {
    JobConf job = new JobConf();
    ...
    LinkedHashMap<String, String> input = new LinkedHashMap<String, String>();
    input.put("pt", "123456");
    InputUtils.addTable(TableInfo.builder().tableName("input_table").partSpec(input).build(), job);
    LinkedHashMap<String, String> output = new LinkedHashMap<String, String>();
    output.put("ds", "654321");
    OutputUtils.addTable(TableInfo.builder().tableName("output_table").partSpec(output).build(), job);
};
    JobClient.runJob(job);
}
```

Example 2:

```

package com.aliyun.odps.mapred.open.example;
...
public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: WordCount <in_table> <out_table>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(SumCombiner.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    Account account = new AliyunAccount("my_access_id", "my_access_key");
    Odps odps = new Odps(account);
    odps.setEndpoint("odps_endpoint_url");
    odps.setDefaultProject("my_project");
    Table table = odps.tables().get(tblname);
    TableInfoBuilder builder = TableInfo.builder().tableName(tblname);
    for (Partition p : table.getPartitions()) {
        if (applicable(p) {
            LinkedHashMap<String, String> partSpec = new LinkedHashMap<String, String>();
            for (String key : p.getPartitionSpec().keys()) {
                partSpec.put(key, p.getPartitionSpec().get(key));
            }
            InputUtils.addTable(builder.partSpec(partSpec).build(), conf);
        }
    }
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    JobClient.runJob(job);
}

```

Note

- In example 2, the MaxCompute SDK and MapReduce SDK are combined to implement a MapReduce task that reads data from specific partitions.
- The code in the example cannot be compiled for execution. It is only an example of the main function.
- The applicable function is user logic that determines whether the partition can be used as the input of a MapReduce job.

8.6.14. Pipeline example

This topic provides a Pipeline example.

Example:

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.Job;

```

```
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.pipeline.Pipeline;
public class WordCountPipelineTest {
    public static class TokenizerMapper extends MapperBase {
        Record word;
        Record one;
        @Override
        public void setup(TaskContext context) throws IOException {
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.setBigint(0, 1L);
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                String[] words = record.get(i).toString().split("\\s+");
                for (String w : words) {
                    word.setString(0, w);
                    context.write(word, one);
                }
            }
        }
    }
    public static class SumReducer extends ReducerBase {
        private Record value;
        @Override
        public void setup(TaskContext context) throws IOException {
            value = context.createOutputValueRecord();
        }
        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            long count = 0;
            while (values.hasNext()) {
                Record val = values.next();
                count += (Long) val.get(0);
            }
            value.set(0, count);
            context.write(key, value);
        }
    }
    public static class IdentityReducer extends ReducerBase {
        private Record result;
        @Override
        public void setup(TaskContext context) throws IOException {
            result = context.createOutputRecord();
        }
        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            while (values.hasNext()) {
                result.set(0, key.get(0));
                result.set(1, values.next().get(0));
                context.write(result);
            }
        }
    }
}
```

```
public static void main(String[] args) throws OdpsException {
    if (args.length != 2) {
        System.err.println("Usage: WordCountPipeline <in_table> <out_table>");
        System.exit(2);
    }
    Job job = new Job();
    /**
     * During pipeline construction, if you do not specify OutputKeySortColumns, PartitionColumns, and OutputGroupingColumns for a mapper, the framework uses OutputKey of the mapper as the default values of these parameters.
     */
    Pipeline pipeline = Pipeline.builder()
        .addMapper(TokenizerMapper.class)
        .setOutputKeySchema(
            new Column[] { new Column("word", OdpsType.STRING) })
        .setOutputValueSchema(
            new Column[] { new Column("count", OdpsType.BIGINT) })
        .setOutputKeySortColumns(new String[] { "word" })
        .setPartitionColumns(new String[] { "word" })
        .setOutputGroupingColumns(new String[] { "word" })
        .addReducer(SumReducer.class)
        .setOutputKeySchema(
            new Column[] { new Column("word", OdpsType.STRING) })
        .setOutputValueSchema(
            new Column[] { new Column("count", OdpsType.BIGINT) })
        .addReducer(IdentityReducer.class).createPipeline();
    // Add the pipeline to JobConf. If you want to configure a combiner, use JobConf.
    job.setPipeline(pipeline);
    // Configure the input and output tables.
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // Submit the job and wait for the job to complete.
    job.submit();
    job.waitForCompletion();
    System.exit(job.isSuccessful() == true ? 0 : 1);
}
}
```

9. MaxCompute Graph

9.1. Graph overview

9.1.1. Overview

This topic describes the components of MaxCompute Graph and the supported operations.

MaxCompute Graph is a processing framework designed for iterative graph computing. Graph computing jobs use graphs to build models. A graph is a collection of vertices and edges that have values.

MaxCompute Graph allows you to perform the following operations on graphs:

- Change the value of a vertex or edge.
- Add or remove a vertex.
- Add or remove an edge.

Note

- When you edit a vertex or edge, you must use code to maintain the relationships between vertices and edges.
- A directed graph has the following types of edges.
 - Outgoing edge: a directed edge on which the current vertex is the origin.
 - Incoming edge: a directed edge on which the current vertex is the destination.

MaxCompute Graph performs iterations to edit graphs, enable graphs to evolve, and obtain analysis results. Typical applications include the PageRank, single source shortest path (SSSP) algorithm, and k-means clustering algorithm. You can use SDK for Java provided by MaxCompute Graph to compile graph computing programs.

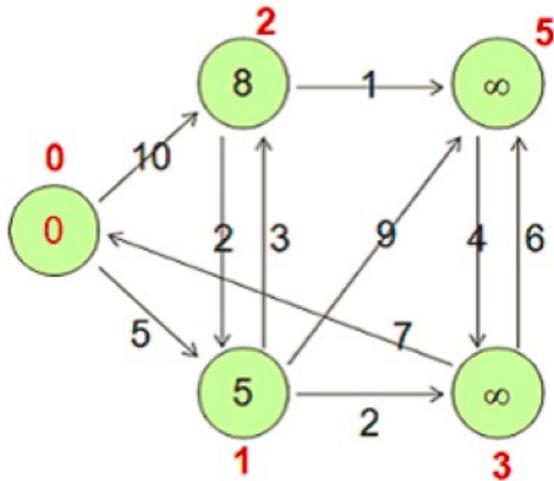
9.1.2. Data structure of MaxCompute Graph

This topic describes the data structure of MaxCompute Graph.

MaxCompute Graph processes directed graphs. MaxCompute provides only a two-dimensional table storage structure. Therefore, you must resolve graph data into two-dimensional tables and store them in MaxCompute.

During graph computing and analytics, use custom GraphLoader to convert two-dimensional table data into vertices and edges that are applicable to MaxCompute Graph. You can determine how to resolve graph data into two-dimensional tables based on your business requirements.

The structure of a vertex is <ID, Value, Halted, Edges>. ID specifies the identifier of the vertex. Value specifies the value of the vertex. Halted specifies whether the iteration of the vertex is terminated. Edges specifies all the edges that start from the vertex. The structure of an edge is <DestVertexID, Value>. DestVertexID specifies the identifier of the destination vertex. Value specifies the value of the edge. The following figure shows the data structure of MaxCompute Graph.



The preceding figure is composed of the vertexes listed in the following table.

Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5>, <2, 10>]>
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1>]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4>]>

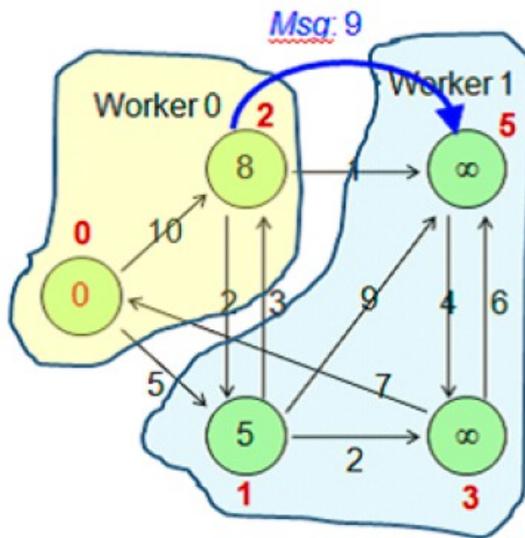
9.1.3. Graph logic

9.1.3.1. Graph loading

A Graph program performs the following operations: graph loading, iterative computing, and iteration termination. This topic describes the graph loading steps in a Graph program.

Graph loading consists of the following steps:

- **Graph loading:** MaxCompute Graph calls the custom GraphLoader to resolve the records in the input table into vertices or edges.
- **Partitioning:** MaxCompute Graph calls the custom Partitioner to partition the vertices and distributes the partitions to the related workers. By default, MaxCompute Graph partitions the vertices based on the hash values of the vertex IDs modulo the number of workers.



In the preceding figure, the number of workers is 2. Vertex 0 and Vertex 2 are distributed to Worker 0 because the result of vertex IDs modulo 2 is 0. Vertex 1, Vertex 3, and Vertex 5 are distributed to Worker 1 because the result of vertex IDs modulo 2 is 1.

9.1.3.2. Iterative computing

A Graph program performs the following operations: graph loading, iterative computing, and iteration termination. This topic describes the iterative computing steps in a Graph program.

An iteration is called a superstep. During an iteration, the program traverses all non-halted vertices or all vertices that receive messages, and calls the `compute(ComputeContext context, Iterable messages)` method. For non-halted vertices, the value of the `Halted` parameter is false. Halted vertices are automatically activated after they receive messages.

The `compute(ComputeContext context, Iterable messages)` method can be used to perform the following operations:

- Process the messages that are sent by the previous superstep to the current vertex.
- Edit a graph as required:
 - Change the values of vertices or edges.
 - Send messages to some vertices.
 - Add or remove vertices or edges.
- Aggregate information to obtain global information by using the aggregator.
- Set the current vertex to the halted or non-halted state.
- Enable MaxCompute Graph to asynchronously send messages to the related workers in each superstep. The messages are then processed in the next superstep.

9.1.3.3. Iteration termination

A Graph program performs the following operations: graph loading, iterative computing, and iteration termination. This topic describes the iteration termination steps in a Graph program.

An iteration ends if one of the following conditions is met:

- All vertices are in the halted state (the value of the `Halted` parameter is true), and no new messages are generated.
- The maximum number of iterations is reached.

- The terminate method of an aggregator returns true.

Example:

```
// 1. load
for each record in input_table { GraphLoader.load();
}
// 2. setup
WorkerComputer.setup();
for each aggr in aggregators { aggr.createStartupValue();
}
for each v in vertices { v.setup();
}
// 3. superstep
for (step = 0; step < max; step ++ ) { for each aggr in aggregators { aggr.createInitialValue();
}
for each v in vertices { v.compute();
}
}
// 4. cleanup
for each v in vertices { v.cleanup();
}
WorkerComputer.cleanup();
```

9.1.4. Aggregator mechanism

This topic describes the implementation mechanism and related API operations of Aggregator. It also describes how to apply Aggregator by using k-means clustering.

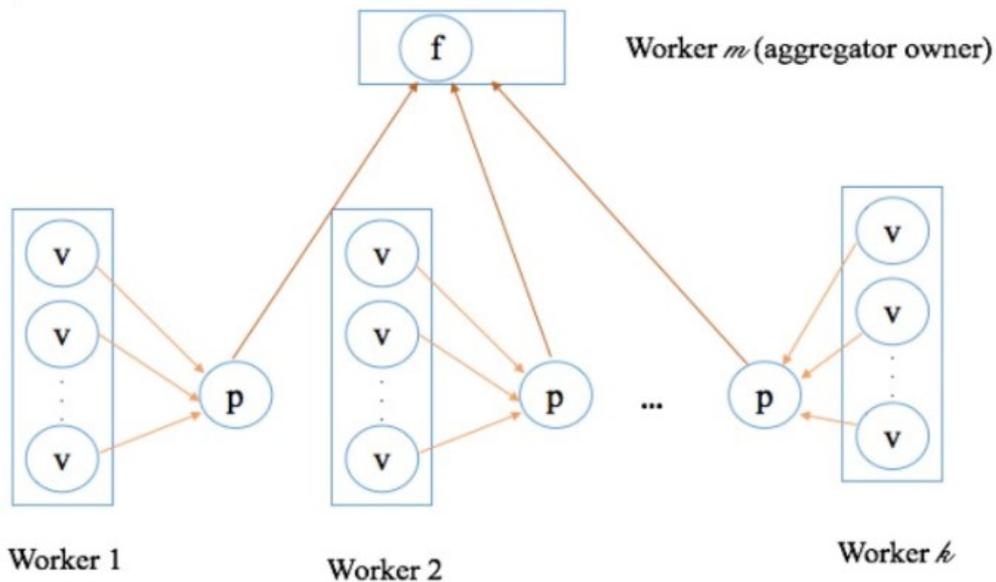
Aggregator is a common feature in MaxCompute Graph jobs and is especially suited for solving machine learning issues. In MaxCompute Graph, Aggregator is used to aggregate and process global information.

Implementation mechanism

The logic of Aggregator is divided into two parts:

- One part is implemented on all workers in distributed mode.
- The other part is only implemented on the worker where Aggregator owner is located in single vertex mode.

Initial values are created and partially aggregated on each worker, and then the partial aggregation results of all workers are sent to the worker where Aggregator owner is located. This worker then aggregates the received partial aggregation objects into a global aggregation result and determines whether to end the iteration. The global aggregation result is distributed to all workers in the next superstep for iteration.



API operations

Aggregator provides five API operations. The following parts describe when to call these API operations and for what purposes.

- `createStartupValue(context)`

This API operation is performed once on all workers before each superstep starts. It is used to initialize `AggregatorValue`. In the first superstep (superstep 0), `WorkerContext.getLastAggregatedValue()` or `ComputeContext.getLastAggregatedValue()` is called to obtain the initialized `AggregatorValue` object.

- `createInitialValue(context)`

This API operation is performed once on all workers when each superstep starts. It is used to initialize `AggregatorValue` for the current iteration. Generally, `WorkerContext.getLastAggregatedValue()` is called to obtain the result of the previous iteration, and then partial initialization is implemented.

- `aggregate(value, item)`

This API operation is also performed on all workers. It is triggered by an explicit call to `ComputeContext#aggregate(item)`, while the preceding two API operations are automatically called by the framework.

This API operation is used to implement partial aggregation. The first parameter `value` indicates the aggregation result of the worker in the current superstep. The initial value is the object returned by `createInitialValue`. The second parameter is passed in when `ComputeContext#aggregate(item)` is called by using user code. In this API operation, `item` is typically used to update value for aggregation. After all the aggregate operations are performed, the obtained value is the partial aggregation result of the worker. The result is then sent by the framework to the worker where Aggregator owner is located.

- `merge(value, partial)`

This API operation is performed on the worker where the Aggregator owner resides. It is used to merge partial aggregation results of workers to obtain the global aggregation object. Similar to `aggregate`, `value` in this API operation indicates the aggregated results, and `partial` indicates objects that you want to aggregate. `partial` is used to update value.

For example, three workers w_0 , w_1 , and w_2 generate partial aggregation results p_0 , p_1 , and p_2 . If p_1 , p_0 , and p_2 are sent in sequence to the worker where the Aggregator owner resides, the merge operations are performed in the following sequence:

- i. $\text{merge}(p_1, p_0)$ is first executed to aggregate p_1 and p_0 as p_1 .
- ii. $\text{merge}(p_1, p_2)$ is executed to aggregate p_1 and p_2 as p_1 . p_1 is the global aggregation result in this superstep.

Therefore, if only one worker exists, the merge method is not required. In this case, $\text{merge}()$ is not called.

- `terminate(context, value)`

After the worker where the Aggregator owner resides executes $\text{merge}()$, the framework calls $\text{terminate}(\text{context}, \text{value})$ to perform the final processing. The second parameter `value` indicates the global aggregation result obtained by calling $\text{merge}()$. The global aggregation result can be further modified in this API operation. After $\text{terminate}()$ is executed, the framework distributes the global aggregation object to all workers for the next superstep.

If `true` is returned for $\text{terminate}()$, iteration is ended for the entire job. Otherwise, the iteration continues. If `true` is returned after convergence is completed, jobs are immediately ended. This applies to machine learning scenarios.

K-means clustering example

This section uses k-means clustering as an example to demonstrate how to use Aggregator.

 **Note** For the complete code, see [Kmeans](#). In this section, the code is resolved and is for reference only.

- GraphLoader

GraphLoader is used to load an input table and convert it to vertices or edges of a graph. In this example, each row of data in the input table is a sample, each sample constructs a vertex, and vertex values are used to store samples.

A Writable class `KmeansValue` is defined as the value type of a vertex.

```
public static class KmeansValue implements Writable {
    DenseVector sample;
    public KmeansValue() {
    }
    public KmeansValue(DenseVector v) {
        this.sample = v;
    }
    @Override
    public void write(DataOutput out) throws IOException {
        writeForDenseVector(out, sample);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        sample = readFieldsForDenseVector(in);
    }
}
```

A `DenseVector` object is encapsulated in `KmeansValue` to store a sample. The `DenseVector` type stems from [matrix-toolkits-java](#). `writeForDenseVector()` and `readFieldsForDenseVector()` are used for serialization and deserialization. For more information, see the complete code.

Custom `KmeansReader` code:

```

public static class KmeansReader extends
    GraphLoader<LongWritable, KmeansValue, NullWritable, NullWritable> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, KmeansValue, NullWritable, NullWritable> context)
        throws IOException {
        KmeansVertex v = new KmeansVertex();
        v.setId(recordNum);
        int n = record.size();
        DenseVector dv = new DenseVector(n);
        for (int i = 0; i < n; i++) {
            dv.set(i, ((DoubleWritable)record.get(i)).get());
        }
        v.setValue(new KmeansValue(dv));
        context.addVertexRequest(v);
    }
}

```

In `KmeansReader`, a vertex is created when each row of data (a record) is read. `recordNum` is used as the vertex ID, and the record content is converted to a `DenseVector` object and encapsulated in `VertexValue`.

- Vertex

Custom `KmeansVertex` code:

```

public static class KmeansVertex extends
    Vertex<LongWritable, KmeansValue, NullWritable, NullWritable> {
    @Override
    public void compute(
        ComputeContext<LongWritable, KmeansValue, NullWritable, NullWritable> context,
        Iterable<NullWritable> messages) throws IOException {
        context.aggregate(getValue());
    }
}

```

The logic of the preceding code is to implement partial aggregation for samples maintained for each iteration. For more information about the logic, see the implementation of `Aggregator` in the following section.

- Aggregator

The main logic of k-means is concentrated on `Aggregator`. Custom `KmeansAggrValue` is used to maintain the content you want to aggregate and distribute.

```

public static class KmeansAggrValue implements Writable {
    DenseMatrix centroids;
    DenseMatrix sums; // used to recalculate new centroids
    DenseVector counts; // used to recalculate new centroids
    @Override
    public void write(DataOutput out) throws IOException {
        writeForDenseDenseMatrix(out, centroids);
        writeForDenseDenseMatrix(out, sums);
        writeForDenseVector(out, counts);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        centroids = readFieldsForDenseMatrix(in);
        sums = readFieldsForDenseMatrix(in);
        counts = readFieldsForDenseVector(in);
    }
}

```

In the preceding code, three objects are maintained in `KmeansAggrValue`:

- `centroids`: indicates the existing K centers. If the sample is m -dimensional, `centroids` is a matrix of $K \times m$.
- `sums`: indicates a matrix of the same size as `centroids`. Each element records the sum of a specific dimension of the sample that is closest to a specific center. For example, `sums(i,j)` indicates the sum of dimension j of the sample that is closest to center i .
- `counts` is a K -dimensional vector. It records the number of samples closest to each center. `counts` is used with `sums` to calculate a new center, which is the main content to be aggregated.

`KmeansAggregator` is a custom `Aggregator` implementation class. The implementation is described below in order of the preceding API operations:

i. Implementation of `createStartupValue()`

```

public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {
    public KmeansAggrValue createStartupValue(WorkerContext context) throws IOException {
        KmeansAggrValue av = new KmeansAggrValue();
        byte[] centers = context.readCacheFile("centers");
        String lines[] = new String(centers).split("\n");
        int rows = lines.length;
        int cols = lines[0].split(",").length; // assumption rows >= 1
        av.centroids = new DenseMatrix(rows, cols);
        av.sums = new DenseMatrix(rows, cols);
        av.sums.zero();
        av.counts = new DenseVector(rows);
        av.counts.zero();
        for (int i = 0; i < lines.length; i++) {
            String[] ss = lines[i].split(",");
            for (int j = 0; j < ss.length; j++) {
                av.centroids.set(i, j, Double.valueOf(ss[j]));
            }
        }
        return av;
    }
}

```

This method initializes a `KmeansAggrValue` object, reads the initial center from the `centers` file, and assigns a value to `centroids`. The initial values of `sums` and `counts` are 0.

ii. Implementation of `createInitialValue()`

```
@Override
public KmeansAggrValue createInitialValue(WorkerContext context)
    throws IOException {
    KmeansAggrValue av = (KmeansAggrValue)context.getLastAggregatedValue(0);
    // reset for next iteration
    av.sums.zero();
    av.counts.zero();
    return av;
}
```

This method first obtains `KmeansAggrValue` of the previous iteration and clears the values of sums and counts. Only the centroids value of the previous iteration is retained.

iii. Implementation of `aggregate()`

```
@Override
public void aggregate(KmeansAggrValue value, Object item)
    throws IOException {
    DenseVector sample = ((KmeansValue)item).sample;
    // find the nearest centroid
    int min = findNearestCentroid(value.centroids, sample);
    // update sum and count
    for (int i = 0; i < sample.size(); i++) {
        value.sums.add(min, i, sample.get(i));
    }
    value.counts.add(min, 1.0d);
}
```

This method calls `findNearestCentroid()` to find the index of the center closest to the sample item, uses sums to add up all dimensions, and increments the value of counts by 1.

The preceding three methods are executed on all workers to implement partial aggregation. The following methods can be used to implement global aggregation on the worker where the Aggregator owner resides:

i. Implementation of `merge()`

```
@Override
public void merge(KmeansAggrValue value, KmeansAggrValue partial)
    throws IOException {
    value.sums.add(partial.sums);
    value.counts.add(partial.counts);
}
```

In the preceding example, the implementation logic of `merge` is to add the values of sums and counts aggregated by each worker.

ii. Implementation of `terminate()`

```

@Override
public boolean terminate(WorkerContext context, KmeansAggrValue value)
    throws IOException {
    // Calculate the new means to be the centroids (original sums)
    DenseMatrix newCentroids = calculateNewCentroids(value.sums, value.counts, value.centroids
);
    // print old centroids and new centroids for debugging
    System.out.println("\nsuperstep: " + context.getSuperstep() +
        "\nold centroid:\n" + value.centroids + " new centroid:\n" + newCentroids);
    boolean converged = isConverged(newCentroids, value.centroids, 0.05d);
    System.out.println("superstep: " + context.getSuperstep() + "/"
        + (context.getMaxIteration() - 1) + " converged: " + converged);
    if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
        // converged or reach max iteration, output centroids
        for (int i = 0; i < newCentroids.numRows(); i++) {
            Writable[] centroid = new Writable[newCentroids.numColumns()];
            for (int j = 0; j < newCentroids.numColumns(); j++) {
                centroid[j] = new DoubleWritable(newCentroids.get(i, j));
            }
            context.write(centroid);
        }
        // true means to terminate iteration
        return true;
    }
    // update centroids
    value.centroids.set(newCentroids);
    // false means to continue iteration
    return false;
}

```

In the preceding example, `terminate()` calls `calculateNewCentroids()` based on sums and counts to calculate the average value and obtain a new center. Then, `isConverged()` is called to check whether the center is converged based on the Euclidean distance between the new and old centers. If the number of convergences or iterations reaches the upper limit, the new center is generated, and `true` is returned to end the iteration. Otherwise, the center is updated, and `false` is returned to continue the iteration.

iii. main method

The main method is used to construct `GraphJob`, configure related settings, and submit a job.

```

public static void main(String[] args) throws IOException {
    if (args.length < 2)
        printUsage();
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(KmeansReader.class);
    job.setRuntimePartitioning(false);
    job.setVertexClass(KmeansVertex.class);
    job.setAggregatorClass(KmeansAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
        job.setMaxIteration(Integer.parseInt(args[2]));
    long start = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}

```

Note If `job.setRuntimePartitioning` is set to `false`, data loaded by each worker is not partitioned based on the partitioner. Data is loaded and maintained by the same worker.

Summary

Basic steps to implement Aggregator:

1. When each worker starts, it executes `createStartUpValue` to create `AggregatorValue`.
2. Before each iteration starts, each worker executes `createInitialValue` to initialize `AggregatorValue` for the iteration.
3. In an iteration, each vertex uses `context.aggregate()` to call `aggregate()` to implement partial iteration in the worker.
4. Each worker sends the partial iteration result to the worker where the Aggregator owner resides.
5. The worker where the Aggregator owner resides executes `merge` multiple times to implement global aggregation.
6. The worker where the Aggregator owner resides executes `terminate` to process the global aggregation result and determines whether to end the iteration.

9.2. Graph feature overview

9.2.1. Run a job

This topic describes how to run a MaxCompute Graph job.

The MaxCompute client provides a JAR command for you to run MaxCompute Graph jobs. This command is used in the same way as the JAR command in MapReduce.

Syntax:

```

Usage: jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS]
  -conf <configuration_file>      Specify an application configuration file
  -classpath <local_file_list>    classpaths used to run mainClass
  -D <name>=<value>               Property value pair, which will be used to run mainClass
  -local                           Run job in local mode
  -resources <resource_name_list> file/table resources used in graph, seperate by comma

```

The following table describes the parameters in this command.

Parameters

Parameter	Description
-conf <configuration file>	The JobConf configuration file.
-classpath <local_file_list>	The classpath used to run a job in local mode. This parameter specifies the local paths of the JAR package where the Main function is located. The paths include both the relative and absolute paths.
-D <prop_name>=<prop_value>	The Java property of <mainClass> for local execution. You can specify multiple properties.
-local	Specifies that the Graph jobs run in local mode. It is used for program debugging.
-resources <resource_name_list>	<p>Declares the resources used for running the Graph job. In most cases, you must specify the names of the resources that the Graph job uses in resource_name_list.</p> <div style="border: 1px solid #add8e6; padding: 5px; background-color: #e6f2ff;"> <p> Note If you read other MaxCompute resources when you run the Graph job, you must add the names of those resources to <resource_name_list>. Multiple resources must be separated by commas (.). If you use cross-project resources, add PROJECT_NAME/resources/ before <resource_name_list>. Example: -resources otherproject/resources/resfile.</p> </div>

 **Note** The preceding optional parameters are included in <GENERIC_OPTIONS>.

You can directly run the main function in the Graph job to submit the job to MaxCompute, instead of submitting the job by using the MaxCompute client. The PageRank algorithm is used in the following example:

```

public static void main(String[] args) throws IOException {
    if (args.length < 2)
        printUsage();
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(PageRankVertexReader.class);
    job.setVertexClass(PageRankVertex.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // Add the resources used in the job to the cache resource. These resources correspond to those specified by -resources and -libjars in the jar command.
    job.addCacheResource("mapreduce-examples.jar");
    // Add the used JAR file and other files to the class cache resource. These files correspond to those specified by -libjars in the JAR command.
    job.addCacheResourceToClassPath("mapreduce-examples.jar");
    // Set the configuration item that corresponds to odps_config.ini in the client. Replace it with the actual one in your configuration file.
    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setDefaultProject(project);
    odps.setEndpoint(endpoint);
    SessionState.get().setOdps(odps);
    SessionState.get().setLocalRun(false); // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
        job.setMaxIteration(Integer.parseInt(args[2]));
    long startTime = System.currentTimeMillis(); job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}

```

9.2.2. Input and output

This topic describes how to define the input and output of a MaxCompute Graph job.

The input and output of MaxCompute Graph jobs must be tables. You cannot customize the input or output format.

Multiple tables and partitions can be used as input. Example of job input definition:

```

GraphJob job = new GraphJob();
job.addInput(TableInfo.builder().tableName("tblname").build());
// Tables are used as input.
job.addInput(TableInfo.builder().tableName("tblname").partSpec("pt1=a/pt2=b").build());
// Partitions are used as input.
job.addInput(TableInfo.builder().tableName("tblname").partSpec("pt1=a/pt2=b").build(), new String[]{"col2", "col0 "});
// Read only the col2 and col0 columns of the input table. Use record.get(0) to obtain col2 in the load() method of GraphLoader. Both are read in the same sequence.

```

Note

- Partition filtering is not supported.
- The addInput framework reads records from the input table and transfers the records to the user-defined GraphLoader to load graph data.

Multiple tables and partitions can be used as output, and each of them must be marked with a label. Example of job output definition:

```
GraphJob job = new GraphJob();
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").build());
// If the output table is a partitioned table, the last level of partitions must be provided.
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").label("output1").build(), true);
// true indicates that the code overwrites partitions specified by tableinfo. The value true is similar to the INSERT OVERWRITE statement. The value false is similar to the INSERT INTO statement.
```

Note A Graph job can use the `write()` method of `WorkContext` to write data to an output table during runtime. If you want to write data to multiple tables, you must mark each table with a label, such as `output1` in the preceding example.

9.2.3. Read data from resources

9.2.3.1. Use GraphJob to specify resources to be read

This topic describes how to use `GraphJob` to read resources in MaxCompute Graph.

In addition to the JAR command, you can use the following two methods of `GraphJob` to specify the resources to be read by Graph:

```
void addCacheResources(String resourceNames)
void addCacheResourcesToClassPath(String resourceNames)
```

9.2.3.2. Use resources in Graph

This topic describes how to use `WorkerContext` to read resources in MaxCompute Graph.

You can use a `WorkerContext` object to read resources in MaxCompute Graph.

```
public byte[] readCacheFile(String resourceName) throws IOException;
public Iterable<byte[]> readCacheArchive(String resourceName) throws IOException;
public Iterable<byte[]> readCacheArchive(String resourceName, String relativePath) throws IOException;
;
public Iterable<WritableRecord> readResourceTable(String resourceName);
public BufferedInputStream readCacheFileAsStream(String resourceName) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName) throws IOException;
n;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName, String relativePath) throws IOException;
```

Note

- You can use the `setup()` method of `WorkerComputer` to read resources and save the resources in `WorkerValue`. Then, you can use `getWorkerValue` to obtain the resources.
- If you want to read resources and process them at the same time, you can use the preceding stream APIs. This method reduces memory consumption.

9.3. Graph SDK

This topic describes the commonly used classes of Graph SDK.

Class	Description
GraphJob	Defines, submits, and manages a MaxCompute Graph job. The class inherits JobConf.
Vertex	Defines a vertex in a graph. A vertex has the following properties: id, value, halted, and edges. You can specify this class by using the setVertexClass method of GraphJob.
Edge	Defines an edge in a graph. An edge has the following properties: destVertexId and value. MaxCompute Graph uses the adjacency list as the data structure. The outgoing edges of a vertex are specified by the edges property of the vertex.
GraphLoader	Loads a graph. You can specify this class by using the setGraphLoaderClass method of GraphJob.
VertexResolver	Customizes the logic for handling conflicts during graph topology modification. You can specify this class by using the setLoadingVertexResolverClass and setComputingVertexResolverClass methods of GraphJob. The setLoadingVertexResolverClass method specifies the class that is used during graph loading, whereas the setComputingVertexResolverClass method specifies the class that is used during iterative computing.
Partitioner	Specifies how to distribute vertices to workers. This way, computing can be performed by multiple workers at the same time. You can specify this class by using the setPartitionerClass method of GraphJob. By default, the HashPartitioner class is used. HashPartitioner computes the hash value of a vertex ID and divides the hash value by the number of workers to obtain the remainder of the hash value. This remainder specifies the worker to which the vertex is distributed.
WorkerComputer	Customizes the operations to perform when a worker starts and stops. You can specify this class by using the setWorkerComputerClass method of GraphJob.
Aggregator	Processes and summarizes global information. You can specify one or more Aggregator classes by using the setAggregatorClass(Class...) method of GraphJob.
Combiner	Summarizes the output records. You can specify this class by using the setCombinerClass method of GraphJob.
Counters	Defines a counter that is used for counting workers. In the operating logic of a job, you can use the WorkerContext class to obtain the counter and count workers. The framework then automatically calculates the sum of the workers.
WorkerContext	Defines the context that encapsulates the features provided by the framework, such as the features that allow workers to modify the graph topology, send messages, write results, and read resources.

9.4. Development and debugging

9.4.1. Development process

This topic describes the development process of MaxCompute Graph.

Graph development plug-ins are unavailable in MaxCompute. You can use Eclipse to develop MaxCompute Graph programs. We recommend that you perform the following steps to develop a MaxCompute Graph program:

1. Compile Graph code and perform local debugging to test basic functions.
2. Perform cluster debugging to verify the result.

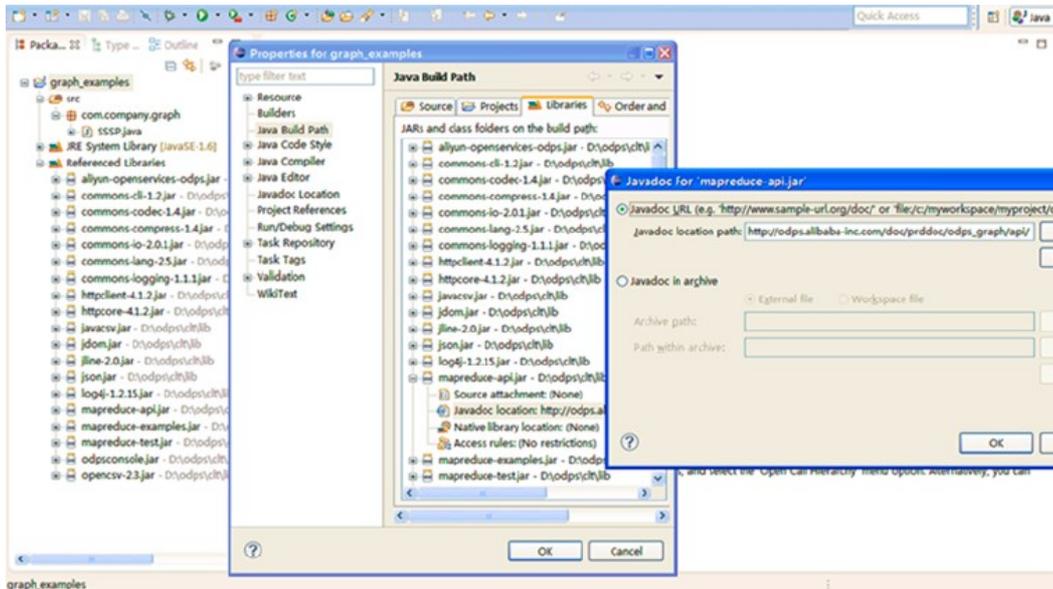
9.4.2. Development example

This topic uses the single source shortest path (SSSP) algorithm as an example to describe how to use Eclipse to develop and debug a MaxCompute Graph program.

Procedure

1. Create a Java project named graph_examples.
2. Add the JAR package in the lib directory on the MaxCompute client to Java Build Path of the Eclipse project.

The following figure shows a configured Eclipse project.



3. Develop a MaxCompute Graph program.
 In the actual development process, an example program, such as SSSP, is first copied and then modified. In this example, only the package path is changed to *package com.aliyun.odps.graph.example*.
4. Compile and package the code. In an Eclipse environment, right-click the source code directory (the src directory in the preceding figure) and choose **Export > Java > JAR file** to generate a JAR package. Then, select the path to store the JAR package, such as *D:\odps\clt\odps-graph-example-sssp.jar*.
5. Use the MaxCompute client to run SSSP. For more information, see [Compile and run a Graph job](#).

9.4.3. Perform debugging on the local computer

MaxCompute Graph supports the local debugging mode. This topic describes how to use Eclipse for breakpoint debugging.

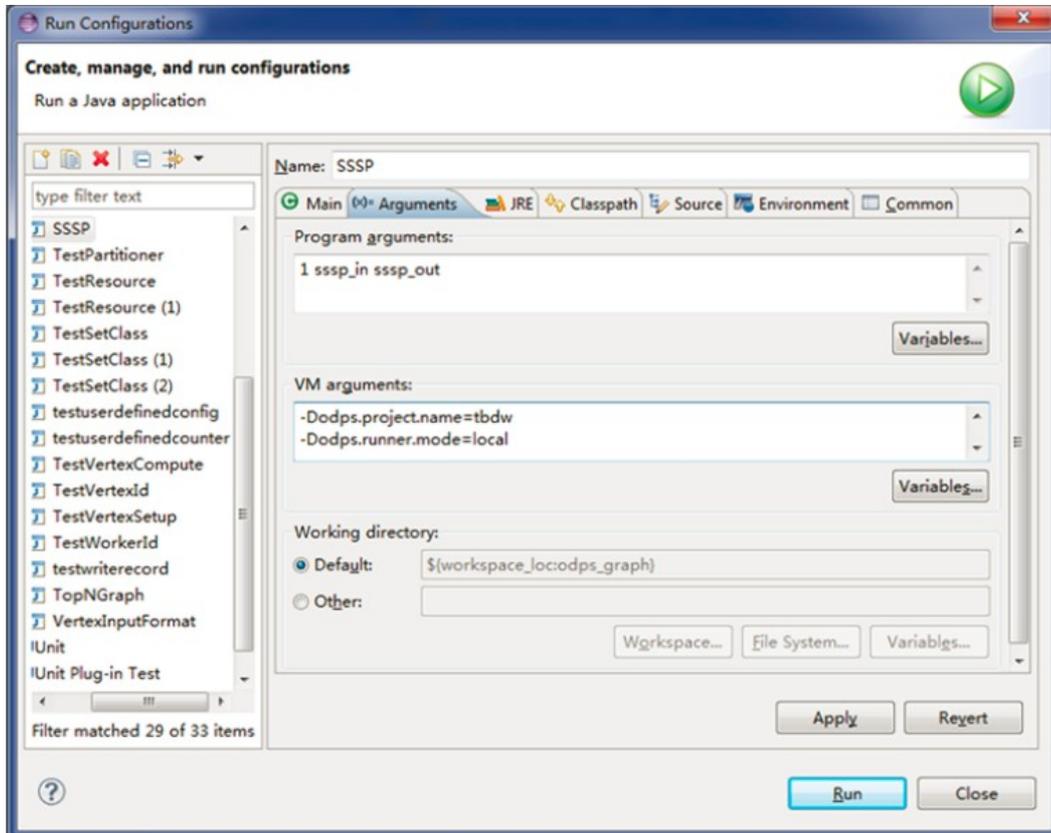
Procedure

1. Download a [Maven package](#) named odps-graph-local.
2. Select the Eclipse project, right-click the main program file that contains the main function of a Graph job, and choose **Run As > Run Configurations** to configure parameters.
3. On the **Arguments** tab, set Program arguments to `1 sssp_in sssp_out` as the input parameter of the main program.
4. On the **Arguments** tab, set VM arguments to the following content:

```

-Dodps.runner.mode=local
-Dodps.project.name=<project.name>
-Dodps.end.point=<end.point>
-Dodps.access.id=<access.id>
-Dodps.access.key=<access.key>

```



- In local mode in which `odps.end.point` is not specified, create the `sssp_in` and `sssp_out` tables in the warehouse directory and add the following data to the `sssp_in` table:

```

1, "2:2,3:1,4:4"
2, "1:2,3:2,4:1"
3, "1:1,2:2,5:1"
4, "1:4,2:1,5:1"
5, "3:1,4:1"

```

Note For more information about the warehouse directory, see [Run MapReduce tasks locally](#).

- Click **Run** to run SSSP on the local machine.

Note

Configure the parameters based on the settings in `conf/odps_config.ini` on the MaxCompute client. The preceding parameters are commonly used. Descriptions of other parameters:

- `odps.runner.mode`: The value is `local`. This parameter is required for the local debugging feature.
- `odps.project.name`: specifies the current project. This parameter is required.
- `odps.end.point`: specifies the endpoint of MaxCompute. This parameter is optional. If this parameter is not specified, metadata and data of tables or resources are only read from the warehouse directory. An exception is reported if such data does not exist in the directory. If this parameter is specified, metadata and data are read from the warehouse directory first and then from the remote MaxCompute server if such data does not exist in the directory.
- `odps.access.id`: specifies the AccessKey ID used to access MaxCompute. This parameter is valid only if `odps.end.point` is specified.
- `odps.access.key`: specifies the AccessKey secret used to access MaxCompute. This parameter is valid only if `odps.end.point` is specified.
- `odps.cache.resources`: specifies the resources you want to use. This parameter functions the same as `-resources` of the `jar` command.
- `odps.local.warehouse`: specifies the local path to warehouse. The default value is `./warehouse`.

Output of the local SSSP debugging in Eclipse:

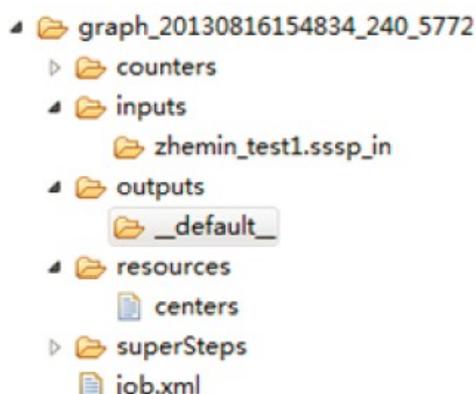
```
Counters: 3
    com.aliyun.odps.graph.local.COUNTER
        TASK_INPUT_BYTE=211
        TASK_INPUT_RECORD=5
        TASK_OUTPUT_BYTE=161
        TASK_OUTPUT_RECORD=5
graph task finish
```

 **Notice** In this example, the `sssp_in` and `sssp_out` tables must exist in the local warehouse directory. For more information about the `sssp_in` and `sssp_out` tables, see [Compile and run a Graph job](#).

9.4.4. Temporary directory of a local job

This topic describes the temporary directory created when MaxCompute Graph runs a local debugging job.

Each time MaxCompute Graph runs a local debugging job, it creates a temporary directory in the Eclipse project directory, as shown in the following figure.



The temporary directory of a local Graph job contains the following directories and files:

- **counters:** stores the counter information that is generated during the job running.
- **inputs:** stores the input data of the job. Input data is read from the local warehouse directory first. If no data is available, MaxCompute SDK reads data from the server if `odps.end.point` is specified. The `-Dodps.mapred.local.record.limit` parameter specifies the number of records that can be read during each input operation. The default value is 10. The maximum value is 10000.
- **outputs:** stores the output data of the job. If an output table exists in the local warehouse, the results in outputs will overwrite data in that table after the job is executed.
- **resources:** stores the resources that are used by the job. Similar to input data, resource data is read from the local warehouse directory first. If no data is available, MaxCompute SDK reads data from the server if `odps.end.point` is specified.
- **job.xml:** stores job configurations.
- **superstep:** stores persistent messages from each iteration.

 **Note** If you want to produce detailed logs during local debugging, place the `log4j` configuration file named `log4j.properties_odps_graph_cluster_debug` in the `src` directory.

9.4.5. Cluster debugging

This topic describes how to submit a job to a cluster for testing.

After local debugging, you can submit the job to a cluster for testing.

1. Configure the MaxCompute client.
2. Run the `add jar /path/work.jar -f;` command to update the JAR package.
3. Run a `jar` command to run a job. Then, check the operational log and command output.

 **Note** For more information about how to run a Graph job in a cluster, see [Compile and run a Graph job](#).

9.4.6. Performance optimization

This topic describes how to configure job parameters to optimize the performance of Graph.

The following table describes the job parameters that affect the performance of Graph.

Parameter	Description
<code>setSplitSize(long)</code>	The split size of an input table. The unit is MB. The value must be greater than 0. The default value is 64.
<code>setNumWorkers(int)</code>	The number of workers for a job. The value ranges from 1 to 1000. The default value is 1. The number of workers is determined by the input bytes of the job and split size.
<code>setWorkerCPU(int)</code>	The number of CPU resources for a map job. 100 resources are equivalent to one CPU core. The value ranges from 50 to 800. The default value is 200.
<code>setWorkerMemory(int)</code>	The size of memory resources for a map job. The unit is MB. The value ranges from 256 to 12288. The default value is 4096.
<code>setMaxIteration(int)</code>	The maximum number of iterations. The default value is -1. If the value is less than or equal to 0, the job does not stop when the maximum number of iterations is reached.

Parameter	Description
setJobPriority(int)	The job priority. The value ranges from 0 to 9. The default value is 9. A greater value indicates a lower priority.

We recommend that you optimize the performance by using one or more of the following methods:

1. Use `setNumWorkers` to increase the number of workers.
2. Use `setSplitSize` to reduce the split size and increase the data loading speed.
3. Increase the CPU or memory resources for workers.
4. Set the maximum number of iterations. For applications that do not require high precision, you can reduce the number of iterations to accelerate the execution process.

`setNumWorkers` and `setSplitSize` can be used together to accelerate data loading. Assume that the value of `setNumWorkers` equals that of `workerNum`, the value of `setSplitSize` equals that of `splitSize`, and the total number of input bytes is the value of `inputSize`. The number of split data records is calculated by using the following formula: $\text{splitNum} = \text{inputSize} / \text{splitSize}$. Relationship between `workerNum` and `splitNum`:

- If the value of `splitNum` is equal to that of `workerNum`, each worker loads one split data record.
- If the value of `splitNum` is greater than that of `workerNum`, each worker loads one or more split data records.
- If the value of `splitNum` is less than that of `workerNum`, each worker uploads one or no split data record.

Therefore, you can adjust the settings of `workerNum` and `splitNum` to obtain a suitable data loading speed. In the first two cases, data is loaded faster. In the iteration phase, you need only to adjust the setting of `workerNum`. If you set `runtime partitioning` to `False`, we recommend that you use `setSplitSize` to adjust the number of workers or make sure that the conditions in the first two cases are met. In the third case, some workers do not load split data records. Therefore, you can insert `set odps.graph.split.size=<m>; set odps.graph.worker.num=<n>;` before the JAR command to achieve the same effect as `setNumWorkers` and `setSplitSize`.

Another common performance issue is data skew. As indicated by counters, some workers process much more split data records or edges than other workers.

Data skew occurs when the number of split data records, edges, or messages that correspond to some keys is much greater than that of other keys. These keys are processed by a small number of workers, which results in longer runtime. To address this issue, use one or more of the following methods:

- Use a combiner to aggregate the messages of the split data records that correspond to the keys to reduce the number of messages generated.

Developers can define a combiner to reduce the memory and network traffic consumed by message storage, which shortens the job execution duration.

- Improve the business logic.

If the data volume is large, reading data in a disk may take up the processing time. Therefore, you can reduce the data bytes to be read to increase the overall throughput. This improves job performance. To reduce data bytes, use one of the following methods:

- Reduce data input: For some decision-making applications, processing sampled data affects only the precision of the results, not the overall accuracy. In this case, you can perform special data sampling and import the data to the input table for processing.
- Avoid reading fields that are not used: The `TableInfo` class of the MaxCompute Graph framework supports reading specific columns that are transferred by using column name arrays, rather than reading the entire table or partition. This reduces the input data volume and improves job performance.

9.4.7. Built-in JAR packages

This topic describes the built-in JAR packages that are loaded to a Java Virtual Machine (JVM) by default.

By default, the following JAR packages are loaded to a JVM that runs Graph programs. You do not need to manually upload these resources or use `-libjars` to specify them in a command:

- commons-codec-1.3.jar
- commons-io-2.0.1.jar
- commons-lang-2.5.jar
- commons-logging-1.0.4.jar
- commons-logging-api-1.0.4.jar
- guava-14.0.jar
- json.jar
- log4j-1.2.15.jar
- slf4j-api-1.4.3.jar
- slf4j-log4j12-1.4.3.jar
- xmlenc-0.52.jar

 **Note** In the classpath of the JVM, the preceding built-in JAR packages are placed before your JAR packages, which may result in a version conflict. For example, your program calls a specific class function in commons-codec-1.5.jar, but the function is not included in commons-codec-1.3.jar. In this case, you can call a similar function in commons-codec-1.3.jar or wait until MaxCompute is updated to the required version.

9.5. Limits

This topic describes the limits of MaxCompute Graph.

- Each job can reference a maximum 256 resources. Each table or archive is considered as one unit.
- The total size of resources referenced by a job cannot exceed 512 MB.
- The number of the inputs of a job cannot exceed 1,024. The number of input tables cannot exceed 64. The number of the outputs of a job cannot exceed 256.
- Labels that are specified for outputs cannot be null or empty strings. A label cannot exceed 256 characters in length and can contain only letters, digits, underscores (`_`), number signs (`#`), periods (`.`), and hyphens (`-`).
- The number of custom counters in a job cannot exceed 64. group name and counter name of counters cannot contain number signs (`#`). The total length of the two names cannot exceed 100 characters.
- The number of workers for a job is calculated by the framework. The maximum number of workers is 1,000. If the number of workers exceeds this value, an error is reported.
- By default, a worker consumes 200 CPU resources. 100 resources are equivalent to one CPU core. The number of CPU resources consumed by a worker ranges from 50 to 800.
- By default, a worker consumes 4,096 MB memory. The memory consumed by a worker ranges from 256 MB to 12,288 MB.
- A worker can repeatedly read a resource for a maximum of 64 times.
- The default value of split size is 64 MB. You can set the value based on your requirements. The value of this parameter must be greater than 0 and less than or equal to the result of the `9223372036854775807 >> 20` operation.
- GraphLoader, Vertex, and Aggregator in MaxCompute Graph are limited by the Java sandbox when they are run in a cluster. However, the main program of Graph jobs is not limited by the Java sandbox. For more information, see [Java sandbox limits](#).

9.6. Sample programs

9.6.1. SSSP

This topic provides an example of Single Source Shortest Path (SSSP).

Dijkstra's algorithm is a common algorithm that is used to calculate the SSSP in a directed graph.

How the Dijkstra's algorithm works:

- Initialization: The path from s to s is 0 ($d[s] = 0$), and the path from u to s is infinite ($d[u] = \infty$).
- Iteration: If an edge from u to v exists, the shortest path from s to v is updated to $d[v] = \min(d[v], d[u] + \text{weight}(u, v))$. The iteration does not end until the paths from all vertices to s do not change.

Note Shortest path: For a weighted directed graph $G = (V, E)$, multiple paths are available from source vertex s to sink vertex v . The path with the smallest sum of edge weights is called the shortest path from s to v .

The working mode of the algorithm shows that the algorithm is suitable for MaxCompute Graph. Each vertex maintains the current shortest path to the source vertex. If the path changes, the new path is added with the edge weight, and a message is sent to notify adjacent vertices. In the next iteration, the adjacent vertices update the shortest paths based on the received message. If the shortest path between each vertex and the source vertex does not change, the iteration ends.

Example:

```
import java.io.IOException;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.data.TableInfo;
public class SSSP {
    public static final String START_VERTEX = "sssp.start.vertex.id";
    /**Define SSSPVertex, where:
    * The vertex value indicates the current shortest path from this vertex to the source vertex startVertexId.
    * The compute() method uses the iteration formula  $d[v] = \min(d[v], d[u] + \text{weight}(u, v))$  to update the vertex value.
    * The cleanup() method writes the vertex and its shortest path to the source vertex to the result table.
    */
    public static class SSSPVertex extends
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
        private static long startVertexId = -1;
        public SSSPVertex() {
            this.setValue(new LongWritable(Long.MAX_VALUE));
        }
        public boolean isStartVertex(
            ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context) {
            if (startVertexId == -1) {
                String s = context.getConfiguration().get(START_VERTEX);
                startVertexId = Long.parseLong(s);
            }
            return getId().get() == startVertexId;
        }
        @Override
        public void compute(
```

```

ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context, Iterable<LongWritabl
e> messages) throws IOException {
    long minDist = isStartVertex(context) ? 0 : Integer.MAX_VALUE;
    for (LongWritable msg : messages) { if (msg.get() < minDist) {
        minDist = msg.get();
    }
    }
    if (minDist < this.getValue().get()) {
        this.setValue(new LongWritable(minDist));
        if (hasEdges()) {
            for (Edge<LongWritable, LongWritable> e : this.getEdges()) {
                context.sendMessage(e.getDestVertexId(), new LongWritable(minDist + e.getValue().get()));
            }
        }
    } else {
        voteToHalt();
        // If the vertex value does not change, voteToHalt() is called to notify the framework that this ver
        tex enters the halted state. The calculation ends when all vertices enter the halted state.
    }
    }
    @Override
    public void cleanup(
        WorkerContext<LongWritable, LongWritable, LongWritable, LongWritable> context) throws IOException {
        context.write(getId(), getValue());
    }
    }
    /** Define MinLongCombiner and combine messages sent to the same vertex to optimize performance and
    reduce memory usage. */
    public static class MinLongCombiner extends
        Combiner<LongWritable, LongWritable> {
        @Override
        public void combine(LongWritable vertexId, LongWritable combinedMessage, LongWritable messageToCombi
        ne) throws IOException {
            if (combinedMessage.get() > messageToCombine.get()) {
                combinedMessage.set(messageToCombine.get());
            }
        }
    }
    }
    /** Define the SSSPVertexReader class, load a graph, and parse each record in the table into a verte
    x. The first column of the record is the vertex ID, and the second column stores all edge sets start
    ing from the vertex, such as 2:2,3:1,4:4.**/
    public static class SSSPVertexReader extends
        GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
        @Override
        public void load(LongWritable recordNum, WritableRecord record,
            MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context) throws IOException
        {
            SSSPVertex vertex = new SSSPVertex();
            vertex.setId((LongWritable) record.get(0));
            String[] edges = record.get(1).toString().split(",");
            for (int i = 0; i < edges.length; i++) {
                String[] ss = edges[i].split(":");
                vertex.addEdge(new LongWritable(Long.parseLong(ss[0])), new LongWritable(Long.parseLong(ss[1])));
            }
            context.addVertexRequest(vertex);
        }
    }
    }
    public static void main(String[] args) throws IOException { if (args.length < 2) {
        System.out.println("Usage: <startnode> <input> <output>");
    }
}

```

```

System.exit(-1);
}
GraphJob job = new GraphJob();
// Define GraphJob, specify the implementation of Vertex, GraphLoader, and Combiner, and specify input and output tables.
job.setGraphLoaderClass(SSSPVertexReader.class);
job.setVertexClass(SSSPVertex.class);
job.setCombinerClass(MinLongCombiner.class);
job.set(START_VERTEX, args[0]);
job.addInput(TableInfo.builder().tableName(args[1]).build());
job.addOutput(TableInfo.builder().tableName(args[2]).build());
long startTime = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}
}

```

9.6.2. PageRank

This topic provides a PageRank example.

PageRank is a classic algorithm used to rank web pages. The input of the algorithm is a digraph G . Each vertex represents a web page, and the edge between two vertices represents the link between two web pages. Basic principles of the algorithm lie in the following aspects:

- Initialization: A vertex value indicates the rank value of PageRank. Vertex values are of the DOUBLE type. During initialization, the values of all vertices are $1/\text{TotalNumVertices}$.
- Iteration formula: $\text{PageRank}(i) = 0.15/\text{TotalNumVertices} + 0.85 \times \text{sum}$. In this formula, sum indicates the sum of the $\text{PageRank}(j)/\text{out_degree}(j)$ values. j indicates all vertices that point to vertex i .

The PageRank algorithm is suitable for MaxCompute Graph programs. Each vertex j maintains its PageRank value and sends the $\text{PageRank}(j)/\text{out_degree}(j)$ value to its adjacent vertices (for voting) in each iteration. In the next iteration, each vertex recalculates the PageRank value by using the iteration formula.

Example:

```

import java.io.IOException;
import org.apache.log4j.Logger;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
public class PageRank {
private final static Logger LOG = Logger.getLogger(PageRank.class);
/**
 * Define PageRankVertex, where:
 * The vertex value indicates the current PageRank value of the vertex (web page).
 * The compute() method uses the following iteration formula to update the vertex value:  $\text{PageRank}(i) = 0.15/\text{TotalNumVertices} + 0.85 \times \text{sum}$ .
 * The cleanup() method writes the vertex value and its PageRank value to the result table.

```

```

**/
public static class PageRankVertex extends
Vertex<Text, DoubleWritable, NullWritable, DoubleWritable> {
@Override
public void compute(
ComputeContext<Text, DoubleWritable, NullWritable, DoubleWritable> context, Iterable<DoubleWritable>
messages) throws IOException {
if (context.getSuperstep() == 0) {
setValue(new DoubleWritable(1.0 / context.getTotalNumVertices()));
} else if (context.getSuperstep() >= 1) { double sum = 0;
for (DoubleWritable msg : messages) { sum += msg.get();
}
DoubleWritable vertexValue = new DoubleWritable( (0.15f / context.getTotalNumVertices()) + 0.85f * s
um);
setValue(vertexValue);
}
if (hasEdges()) {
context.sendMessageToNeighbors(this, new DoubleWritable(getValue()
.get() / getEdges().size()));
}
}
@Override
public void cleanup(
WorkerContext<Text, DoubleWritable, NullWritable, DoubleWritable> context) throws IOException {
context.write(getId(), getValue());
}
}
/** Define the PageRankVertexReader class, load a graph, and resolve each record in the table into a
vertex. The first column of the table is the source vertices and other columns are the destination v
ertices.**/
public static class PageRankVertexReader extends
GraphLoader<Text, DoubleWritable, NullWritable, DoubleWritable> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<Text, DoubleWritable, NullWritable, DoubleWritable> context) throws IOException {
PageRankVertex vertex = new PageRankVertex();
vertex.setValue(new DoubleWritable(0));
vertex.setId((Text) record.get(0));
System.out.println(record.get(0));
for (int i = 1; i < record.size(); i++) {
Writable edge = record.get(i);
System.out.println(edge.toString());
if (!(edge.equals(NullWritable.get()))) {
vertex.addEdge(new Text(edge.toString()), NullWritable.get());
}
}
LOG.info("vertex eds size: " + (vertex.hasEdges() ? vertex.getEdges().size() : 0));
context.addVertexRequest(vertex);
}
}
private static void printUsage() {
System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
System.exit(-1);
}
public static void main(String[] args) throws IOException { if (args.length < 2)
printUsage();
GraphJob job = new GraphJob();
// Define GraphJob, and specify the implementation method of Vertex/GraphLoader, the maximum number
of iterations (> 30 by default), and the input and output tables.

```

```

job.setGraphLoaderClass(PageRankVertexReader.class);
job.setVertexClass(PageRankVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
// default max iteration is 30
job.setMaxIteration(30); if (args.length >= 3)
job.setMaxIteration(Integer.parseInt(args[2]));
long startTime = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in "
+ (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}
}

```

9.6.3. K-means clustering

This topic provides a k-means clustering example.

K-means clustering is a basic clustering algorithm that is widely used. How k-means clustering works: Clustering is performed around k points in space, and the closest vertices are classified. The values of the clustering centers are updated in sequence by using iterations until the optimal clustering result is obtained.

Procedure to divide the sample set into k classes:

1. Select the initial centers of k classes.
2. In the *i*th iteration, select a sample, calculate its distance to k centers, and then classify the sample into the class of the center with the shortest distance.
3. Use the mean method to update the center value of the class.
4. For all the k centers, if the value remains unchanged or is less than a threshold after the update, the iteration ends. Otherwise, the iteration continues.

Example:

```

import java.io.DataInput; import java.io.DataOutput;
import java.io.IOException;
import org.apache.log4j.Logger;
import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
public class Kmeans {
private final static Logger LOG = Logger.getLogger(Kmeans.class);
/** Define the KmeansVertex class. The compute() method is simple. It calls the aggregate() method of the context object and passes in the value of the current vertex. The value is of the TUPLE type and expressed by vector. */
public static class KmeansVertex extends
Vertex<Text, Tuple, NullWritable, NullWritable> {
@Override

```

```

public void compute(
    ComputeContext<Text, Tuple, NullWritable, NullWritable> context, Iterable<NullWritable> messages) throws IOException { context.aggregate(getValue());
}
}
/** Define the KmeansVertexReader class, load a graph, and parse each record in the table as a vertex. The transmitted value of recordNum is used as the vertex ID. The vertex value is a tuple that consists of all columns in the record. */
public static class KmeansVertexReader extends
    GraphLoader<Text, Tuple, NullWritable, NullWritable> {
    @Override
    public void load(LongWritable recordNum, WritableRecord record, MutationContext<Text, Tuple, NullWritable, NullWritable> context) throws IOException {
        KmeansVertex vertex = new KmeansVertex();
        vertex.setId(new Text(String.valueOf(recordNum.get())));
        vertex.setValue(new Tuple(record.getAll()));
        context.addVertexRequest(vertex);
    }
}
public static class KmeansAggrValue implements Writable {
    Tuple centers = new Tuple();
    Tuple sums = new Tuple();
    Tuple counts = new Tuple();
    @Override
    public void write(DataOutput out) throws IOException {
        centers.write(out);
        sums.write(out); counts.write(out);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        centers = new Tuple();
        centers.readFields(in);
        sums = new Tuple();
        sums.readFields(in);
        counts = new Tuple();
        counts.readFields(in);
    }
    @Override
    public String toString() {
        return "centers " + centers.toString() + ", sums " + sums.toString()
            + ", counts " + counts.toString();
    }
}
/**
 * Define the KmeansAggregator class. This class encapsulates the main logic of the k-means clustering algorithm.
 * createInitialValue is the initial value (the center point for each of the k classes) that is created for each iteration. In the first iteration (superstep 0), the value of this parameter is the initial center point. In other iterations, the value is the new center point when the previous iteration ends.
 * The aggregate() method calculates the distance from each vertex to the centers of different classes, classifies the vertex into the class of the nearest center, and updates sum and count of the class.
 * The merge() method combines sums and counts collected by each worker.
 * The terminate() method calculates a new center point based on sum and count of each class. If the distance between the original and new center points is less than a threshold or the number of iterations reaches the upper limit, the iteration ends, and False is returned. The final center point is written to the result table.
 */

```

```

public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {
    @SuppressWarnings("rawtypes")
    @Override
    public KmeansAggrValue createInitialValue(WorkerContext context)
    throws IOException {
        KmeansAggrValue aggrVal = null;
        if (context.getSuperstep() == 0) {
            aggrVal = new KmeansAggrValue();
            aggrVal.centers = new Tuple();
            aggrVal.sums = new Tuple();
            aggrVal.counts = new Tuple();
            byte[] centers = context.readCacheFile("centers");
            String lines[] = new String(centers).split("\n");
            for (int i = 0;
                i < lines.length; i++) { String[] ss = lines[i].split(",");
                Tuple center = new Tuple();
                Tuple sum = new Tuple();
                for (int j = 0; j < ss.length; ++j) {
                    center.append(new DoubleWritable(Double.valueOf(ss[j].trim())));
                    sum.append(new DoubleWritable(0.0));
                }
                LongWritable count = new LongWritable(0);
                aggrVal.sums.append(sum); aggrVal.counts.append(count);
                aggrVal.centers.append(center);
            }
        } else {
            aggrVal = (KmeansAggrValue) context.getLastAggregatedValue(0);
        }
        return aggrVal;
    }
    @Override
    public void aggregate(KmeansAggrValue value, Object item) {
        int min = 0;
        double mindist = Double.MAX_VALUE;
        Tuple point = (Tuple) item;
        for (int i = 0;
            i < value.centers.size();
            i++) { Tuple center = (Tuple) value.centers.get(i);
            // use Euclidean Distance, no need to calculate sqrt
            double dist = 0.0d;
            for (int j = 0; j < center.size(); j++) {
                double v = ((DoubleWritable) point.get(j)).get()
                    - ((DoubleWritable) center.get(j)).get();
                dist += v * v;
            }
            if (dist < mindist) { mindist = dist; min = i;
            }
        }
        // update sum and count
        Tuple sum = (Tuple) value.sums.get(min);
        for (int i = 0;
            i < point.size(); i++) {
            DoubleWritable s = (DoubleWritable) sum.get(i); s.set(s.get() + ((DoubleWritable) point.get(i)).get(
            ));
        }
        LongWritable count = (LongWritable) value.counts.get(min);
        count.set(count.get() + 1);
    }
    @Override

```

```

public void merge(KmeansAggrValue value, KmeansAggrValue partial) {
    for (int i = 0; i < value.sums.size(); i++) {
        Tuple sum = (Tuple) value.sums.get(i);
        Tuple that = (Tuple) partial.sums.get(i);
        for (int j = 0; j < sum.size(); j++) {
            DoubleWritable s = (DoubleWritable) sum.get(j);
            s.set(s.get() + ((DoubleWritable) that.get(j)).get());
        }
    }
    for (int i = 0; i < value.counts.size(); i++) {
        LongWritable count = (LongWritable) value.counts.get(i);
        count.set(count.get() + ((LongWritable) partial.counts.get(i)).get());
    }
}
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, KmeansAggrValue value) throws IOException {
    // compute new centers
    Tuple newCenters = new Tuple(value.sums.size());
    for (int i = 0; i < value.sums.size(); i++) {
        Tuple sum = (Tuple) value.sums.get(i);
        Tuple newCenter = new Tuple(sum.size());
        LongWritable c = (LongWritable) value.counts.get(i);
        for (int j = 0; j < sum.size(); j++) {
            DoubleWritable s = (DoubleWritable) sum.get(j);
            double val = s.get() / c.get();
            newCenter.set(j, new DoubleWritable(val));
        }
        // reset sum for next iteration
        s.set(0.0d);
    }
    // reset count for next iteration
    c.set(0);
    newCenters.set(i, newCenter);
}
// update centers
Tuple oldCenters = value.centers; value.centers = newCenters;
LOG.info("old centers: " + oldCenters + ", new centers: " + newCenters);
// compare new/old centers
boolean converged = true;
for (int i = 0; i < value.centers.size() && converged; i++) {
    Tuple oldCenter = (Tuple) oldCenters.get(i);
    Tuple newCenter = (Tuple) newCenters.get(i); double sum = 0.0d;
    for (int j = 0; j < newCenter.size(); j++) {
        double v = ((DoubleWritable) newCenter.get(j)).get() - ((DoubleWritable) oldCenter.get(j)).get();
        sum += v * v;
    }
    double dist = Math.sqrt(sum);
    LOG.info("old center: " + oldCenter + ", new center: " + newCenter + ", dist: " + dist);
    // converge threshold for each center: 0.05
    converged = dist < 0.05d;
}
if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
    // converged or reach max iteration, output centers
    for (int i = 0; i < value.centers.size(); i++) { context.write(((Tuple) value.centers.get(i)).toArray());
    }
    // true means to terminate iteration
    return true;
}
}

```

```

// false means to continue iteration
return false;
}
}
private static void printUsage() {
System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
System.exit(-1);
}
/** Define GraphJob, and specify the implementation method of Vertex/GraphLoader/Aggregator, the maximum number of iterations (> 30 by default), and the input and output tables. */
public static void main(String[] args) throws IOException {
if (args.length < 2)
printUsage();
GraphJob job = new GraphJob();
job.setGraphLoaderClass(KmeansVertexReader.class);
job.setRuntimePartitioning(false);
// Define job.setRuntimePartitioning(false). For the k-means clustering algorithm, vertices do not need to be distributed for graph loading. RuntimePartitioning is set to False to improve the performance of graph loading.
job.setVertexClass(KmeansVertex.class);
job.setAggregatorClass(KmeansAggregator.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
// default max iteration is 30
job.setMaxIteration(30); if (args.length >= 3)
job.setMaxIteration(Integer.parseInt(args[2]));
long start = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}
}

```

9.6.4. BiPartiteMatching

This topic provides a BiPartiteMatching example.

A bipartite graph is a graph where vertices are divided into two sets and each edge connects a vertex in one set to a vertex in the other set. In a bipartite graph, a matching is a set of pairwise non-adjacent edges in which no two edges share a common vertex. Bipartite matching is often used for information matching in scenarios with clear supply and demand relationships, such as online dating websites.

Implementation process:

1. From the first vertex on the left, select another unmatched vertex by following an alternating path to find an augmenting path.
2. If the unmatched vertex is found, an augmenting path is found.
3. Update path information, increase the number of matched edges by 1, and stop searching.
4. If no augmenting path is found, begin with another vertex.

Example:

```

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.Random;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;

```

```

import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
public class BipartiteMatching {
private static final Text UNMATCHED = new Text("UNMATCHED");
public static class TextPair implements Writable {
public Text first; public Text second;
public TextPair() { first = new Text();
second = new Text();
}
public TextPair(Text first, Text second) {
this.first = new Text(first);
this.second = new Text(second);
}
@Override
public void write(DataOutput out) throws IOException {
first.write(out);
second.write(out);
}
@Override
public void readFields(DataInput in) throws IOException {
first = new Text();
first.readFields(in);
second = new Text();
second.readFields(in);
}
@Override
public String toString() { return first + ": " + second;
}
}
public static class BipartiteMatchingVertexReader extends
GraphLoader<Text, TextPair, NullWritable, Text> {
@Override
public void load(LongWritable recordNum, WritableRecord record, MutationContext<Text, TextPair, Null
Writable, Text> context) throws IOException {
BipartiteMatchingVertex vertex = new BipartiteMatchingVertex();
vertex.setId((Text) record.get(0));
vertex.setValue(new TextPair(UNMATCHED, (Text) record.get(1)));
String[] adjs = record.get(2).toString().split(",");
for (String adj:adjs) {
vertex.addEdge(new Text(adj), null);
}
context.addVertexRequest(vertex);
}
}
public static class BipartiteMatchingVertex extends
Vertex<Text, TextPair, NullWritable, Text> {
private static final Text LEFT = new Text("LEFT");
private static final Text RIGHT = new Text("RIGHT");
private static Random rand = new Random();
@Override
public void compute(
ComputeContext<Text, TextPair, NullWritable, Text> context, Iterable<Text> messages) throws IOExcept
ion {

```

```
if (isMatched()) { voteToHalt();
return;
}
switch ((int) context.getSuperstep() % 4) {
case 0:
if (isLeft()) {
context.sendMessageToNeighbors(this, getId());
}
break;
case 1:
if (isRight()) {
Text luckyLeft = null;
for (Text message : messages) { if (luckyLeft == null) {
luckyLeft = new Text(message);
} else {
if (rand.nextInt(1) == 0) { luckyLeft.set(message);
}
}
}
if (luckyLeft != null) { context.sendMessage(luckyLeft, getId());
}
}
break;
case 2:
if (isLeft()) {
Text luckyRight = null;
for (Text msg : messages) { if (luckyRight == null) {
luckyRight = new Text(msg);
} else {
if (rand.nextInt(1) == 0) { luckyRight.set(msg);
}
}
}
if (luckyRight != null) {
setMatchVertex(luckyRight);
context.sendMessage(luckyRight, getId());
}
}
break; case 3:
if (isRight()) {
for (Text msg : messages) { setMatchVertex(msg);
}
}
break;
}
}
@Override
public void cleanup(
WorkerContext<Text, TextPair, NullWritable, Text> context) throws IOException {
context.write(getId(), getValue().first);
}
private boolean isMatched() {
return ! getValue().first.equals(UNMATCHED);
}
private boolean isLeft() {
return getValue().second.equals(LEFT);
}
private boolean isRight() {
return getValue().second.equals(RIGHT);
}
```

```

}
private void setMatchVertex(Text matchVertex) { getValue().first.set(matchVertex);
}
}
private static void printUsage() {
System.err.println("BipartiteMatching <input> <output> [maxIteration]");
}
public static void main(String[] args) throws IOException { if (args.length < 2) {
printUsage();
}
GraphJob job = new GraphJob();
job.setGraphLoaderClass(BipartiteMatchingVertexReader.class);
job.setVertexClass(BipartiteMatchingVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
int maxIteration = 30;
if (args.length > 2) {
maxIteration = Integer.parseInt(args[2]);
}
job.setMaxIteration(maxIteration);
job.run();
}
}

```

9.6.5. Strongly connected components

This section provides an example of strongly connected components.

A directed graph is called a strongly connected graph if every vertex is reachable from every other vertex. A strongly connected sub-graph with a large number of vertices in a directed graph is called a strongly connected component.

The algorithm for strongly connected components is based on the parallel coloring algorithm. For more information, see [Optimizing Graph Algorithms on Pregel-like Systems](#). Each vertex contains two fields:

- **colorID**: stores the color of Vertex *v* during forward traversal. At the end of computing, vertices with the same colorID belong to the same strongly connected component.
- **transposeNeighbors**: stores neighbor IDs of Vertex *v* in the transpose graph of the input graph.

The algorithm is implemented in the following steps:

1. **Transpose graph formation**: contains two supersteps. In the first superstep, each vertex sends a message with its ID to all its outgoing neighbors. These IDs are stored in transposeNeighbors in the second superstep.
2. **Trimming**: contains one superstep. Each vertex with only one incoming or outgoing edge sets its colorID to its own ID, and becomes inactive. Subsequent messages sent to these vertices are ignored.
3. **Forward traversal**: contains two subphases (supersteps): **Start** and **Rest**. In the **Start** phase, each vertex sets its colorID to its own ID, and sends the ID to outgoing neighbors. In the **Rest** phase, each vertex uses the maximum colorID it received to update its own colorID, and propagates the colorID until the colorIDs converge. When the colorIDs converge, the master process sets the phase to backward traversal.
4. **Backward traversal**: contains two subphases, **Start** and **Rest**. In the **Start** phase, each vertex whose ID equals its colorID propagates its ID to the vertices in transposeNeighbors and sets its status as inactive. Subsequent messages sent to these vertices are ignored. In each of the **Rest** phase supersteps, each vertex receives a message matching its colorID, propagates its colorID in the transpose graph, and sets its status as inactive. If one or more vertices remain active, the process goes back to the trimming phase.

Example:

```
import java.io.DataInput;
```

```

import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.IntWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
 * Definition from Wikipedia:
 * In the mathematical theory of directed graphs, a graph is said
 * to be strongly connected if every vertex is reachable from every
 * other vertex. The strongly connected components of an arbitrary
 * directed graph form a partition into subgraphs that are themselves
 * strongly connected.
 *
 * Algorithms with four phases as follows.
 * 1. Transpose Graph Formation: Requires two supersteps. In the first
 * superstep, each vertex sends a message with its ID to all its > outgoing
 * neighbors, which in the second superstep are stored > in transposeNeighbors.
 *
 * 2. Trimming: Takes one superstep. Every vertex with only in-coming or
 * only outgoing edges (or neither) sets its colorID to its own ID and
 * becomes inactive. Messages subsequently sent to the vertex > are ignored.
 *
 * 3. Forward-Traversal: There are two sub phases: Start and Rest. In the
 * Start phase, each vertex sets its colorID to its own ID and > propagates
 * its ID to its outgoing neighbors. In the Rest phase, vertices update
 * their own colorIDs with the minimum colorID they have seen, and > propagate
 * their colorIDs, if updated, until the colorIDs converge.
 * Set the phase to Backward-Traversal when the colorIDs converge.
 *
 * 4. Backward-Traversal: We again break the phase into Start and Rest.
 * In Start, every vertex whose ID equals its colorID propagates its ID > to
 * the vertices in transposeNeighbors and sets itself inactive. > Messages
 * subsequently sent to the vertex are ignored. In each of the Rest > phase supersteps,
 * each vertex receiving a message that matches its colorID: (1) > propagates
 * its colorID in the transpose graph; (2) sets itself inactive. > Messages
 * subsequently sent to the vertex are ignored. Set the phase back to Trimming
 * if not all vertex are inactive.
 *
 * http://ilpubs.stanford.edu:8090/1077/3/p535-salihoglu.pdf
 */
public class StronglyConnectedComponents {
    public final static int STAGE_TRANSPOSE_1 = 0;
    public final static int STAGE_TRANSPOSE_2 = 1;
    public final static int STAGE_TRIMMING = 2;
    public final static int STAGE_FW_START = 3;
    public final static int STAGE_FW_REST = 4;
    public final static int STAGE_BW_START = 5;

```

```
public final static int STAGE_BW_REST = 6;
/**
 * The value is composed of component id, incoming neighbors,
 * active status and updated status.
 */
public static class MyValue implements Writable {
    LongWritable sccID;// strongly connected component id
    Tuple inNeighbors; // transpose neighbors
    BooleanWritable active; // vertex is active or not
    BooleanWritable updated; // sccID is updated or not
    public MyValue() {
        this.sccID = new LongWritable(Long.MAX_VALUE);
        this.inNeighbors = new Tuple();
        this.active = new BooleanWritable(true);
        this.updated = new BooleanWritable(false);
    }
    public void setSccID(LongWritable sccID) {
        this.sccID = sccID;
    }
    public LongWritable getSccID() {
        return this.sccID;
    }
    public void setInNeighbors(Tuple inNeighbors) {
        this.inNeighbors = inNeighbors;
    }
    public Tuple getInNeighbors() {
        return this.inNeighbors;
    }
    public void addInNeighbor(LongWritable neighbor) {
        this.inNeighbors.append(new LongWritable(neighbor.get()));
    }
    public boolean isActive() {
        return this.active.get();
    }
    public void setActive(boolean status) {
        this.active.set(status);
    }
    public boolean isUpdated() {
        return this.updated.get();
    }
    public void setUpdated(boolean update) {
        this.updated.set(update);
    }
    @Override
    public void write(DataOutput out) throws IOException {
        this.sccID.write(out);
        this.inNeighbors.write(out);
        this.active.write(out);
        this.updated.write(out);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        this.sccID.readFields(in);
        this.inNeighbors.readFields(in);
        this.active.readFields(in);
        this.updated.readFields(in);
    }
    @Override
    public String toString() {
```

```

StringBuilder sb = new StringBuilder();
sb.append("sccID: " + sccID.get());
sb.append(" inNeighbors: " + inNeighbors.toDelimitedString(','));
sb.append(" active: " + active.get());
sb.append(" updated: " + updated.get());
return sb.toString();
}
}
public static class SCCVertex extends
Vertex<LongWritable, MyValue, NullWritable, LongWritable> {
public SCCVertex() {
this.setValue(new MyValue());
}
@Override
public void compute(
ComputeContext<LongWritable, MyValue, NullWritable, LongWritable> context, Iterable<LongWritable> ms
gs) throws IOException {
// Messages sent to inactive vertex are ignored.
if (! this.getValue().isActive()) {
this.voteToHalt(); return;
}
int stage = ((SCCAggrValue)context.getLastAggregatedValue(0)).getStage(); switch (stage) {
case STAGE_TRANSPOSE_1:
context.sendMessageToNeighbors(this, this.getId());
break;
case STAGE_TRANSPOSE_2:
for (LongWritable msg: msgs) {
this.getValue().addInNeighbor(msg);
}
case STAGE_TRIMMING:
this.getValue().setSccID(getId());
if (this.getValue().getInNeighbors().size() == 0 || this.getNumEdges() == 0) {
this.getValue().setActive(false);
}
break;
case STAGE_FW_START: this.getValue().setSccID(getId());
context.sendMessageToNeighbors(this, this.getValue().getSccID());
break;
case STAGE_FW_REST:
long minSccID = Long.MAX_VALUE;
for (LongWritable msg : msgs) {
if (msg.get() < minSccID) { minSccID = msg.get();
}
}
if (minSccID < this.getValue().getSccID().get()) {
this.getValue().setSccID(new LongWritable(minSccID));
context.sendMessageToNeighbors(this, this.getValue().getSccID());
this.getValue().setUpdated(true);
} else {
this.getValue().setUpdated(false);
}
break;
case STAGE_BW_START:
if (this.getId().equals(this.getValue().getSccID())) {
for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
context.sendMessage((LongWritable)neighbor, this.getValue().getSccID());
}
this.getValue().setActive(false);
}
}
}
}

```

```

break;
case STAGE_BW_REST: this.getValue().setUpdated(false);
for (LongWritable msg : msgs) {
if (msg.equals(this.getValue().getScCID())) {
for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
context.sendMessage((LongWritable)neighbor, this.getValue().getScCID());
}
this.getValue().setActive(false);
this.getValue().setUpdated(true);
break;
}
}
break;
}
context.aggregate(0, getValue());
}
@Override
public void cleanup(
WorkerContext<LongWritable, MyValue, NullWritable, LongWritable> context)
throws IOException {
context.write(getId(), getValue().getScCID());
}
}
/**
 * The SCCAggrValue maintains global stage and graph updated and > active status.
 * updated is true only if one vertex is updated.
 * active is true only if one vertex is active.
 */
public static class SCCAggrValue implements Writable {
IntWritable stage = new IntWritable(STAGE_TRANSPOSE_1);
BooleanWritable updated = new BooleanWritable(false);
BooleanWritable active = new BooleanWritable(false);
public void setStage(int stage) { this.stage.set(stage);
}
public int getStage() { return this.stage.get();
}
public void setUpdated(boolean updated) {
this.updated.set(updated);
}
public boolean getUpdated() {
return this.updated.get();
}
public void setActive(boolean active){
this.active.set(active);
}
public boolean getActive() {
return this.active.get();
}
@Override
public void write(DataOutput out) throws IOException {
this.stage.write(out);
this.updated.write(out);
this.active.write(out);
}
@Override
public void readFields(DataInput in) throws IOException {
this.stage.readFields(in);
this.updated.readFields(in);
this.active.readFields(in);
}
}

```

```

}
}
/**
 * The job of SCCAggregator is to schedule global stage in > every superstep.
 */
public static class SCCAggregator extends Aggregator<SCCAggrValue> {
@SuppressWarnings("rawtypes")
@Override
public SCCAggrValue createStartupValue(WorkerContext context) throws IOException { return new SCCAggrValue();
}
@SuppressWarnings("rawtypes")
@Override
public SCCAggrValue createInitialValue(WorkerContext context) throws IOException {
return (SCCAggrValue) context.getLastAggregatedValue(0);
}
@Override
public void aggregate(SCCAggrValue value, Object item) throws IOException { MyValue v = (MyValue)item;
if ((value.getStage() == STAGE_FW_REST || value.getStage() == STAGE_BW_REST)&& v.isUpdated()) { value.setUpdated(true);
}
// only active vertex invoke aggregate()
value.setActive(true);
}
@Override
public void merge(SCCAggrValue value, SCCAggrValue partial) throws IOException {
boolean updated = value.getUpdated() || partial.getUpdated();
value.setUpdated(updated);
boolean active = value.getActive() || partial.getActive();
value.setActive(active);
}
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, SCCAggrValue value) throws IOException {
// If all vertices is inactive, job is over.
if (! value.getActive()) { return true;
}
// state machine
switch (value.getStage()) {
case STAGE_TRANSPOSE_1:value.setStage(STAGE_TRANSPOSE_2);
break;
case STAGE_TRANSPOSE_2:value.setStage(STAGE_TRIMMING);
break;
case STAGE_TRIMMING:value.setStage(STAGE_FW_START);
break;
case STAGE_FW_START: value.setStage(STAGE_FW_REST);
break;
case STAGE_FW_REST:if (value.getUpdated()) {
value.setStage(STAGE_FW_REST);
} else {
value.setStage(STAGE_BW_START);
}
break;
case STAGE_BW_START: value.setStage(STAGE_BW_REST);
break;
case STAGE_BW_REST:if (value.getUpdated()) { value.setStage(STAGE_BW_REST);
} else { value.setStage(STAGE_TRIMMING);
}
}
}
}

```

```

break;
}
value.setActive(false);
value.setUpdated(false);
return false;
}
}
public static class SCCVertexReader extends
GraphLoader<LongWritable, MyValue, NullWritable, LongWritable> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, MyValue, NullWritable, LongWritable> context) throws IOException {
SCCVertex vertex = new SCCVertex();
vertex.setId((LongWritable) record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) { try {
long destID = Long.parseLong(edges[i]);
vertex.addEdge(new LongWritable(destID), NullWritable.get());
} catch (NumberFormatException nfe) { System.err.println("Ignore " + nfe);
}
}
context.addVertexRequest(vertex);
}
}
public static void main(String[] args) throws IOException {
if (args.length < 2) {
System.out.println("Usage: <input> <output>");
System.exit(-1);
}
GraphJob job = new GraphJob();
job.setGraphLoaderClass(SCCVertexReader.class);
job.setVertexClass(SCCVertex.class);
job.setAggregatorClass(SCCAggregator.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
long startTime = System.currentTimeMillis();
job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + " second
s");
}
}

```

9.6.6. Connected components

This section provides an example of connected components.

Two vertices are connected if a path exists between them. If each vertex in an undirected graph G is connected to all the other vertices in the graph, G is called a connected graph. Otherwise, G is called an unconnected graph. A connected sub-graph with a large number of vertices is called a connected component.

This algorithm calculates connected component members of each vertex, and generates the connected component of the vertex value that includes the smallest vertex ID. The smallest vertex ID is propagated along edges to all vertices of the connected component.

Example:

```

import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;

```

```

import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.examples.SSSP.MinLongCombiner;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.WritableRecord;
/**
 * Compute the connected component membership of each vertex and output
 * each vertex which's value containing the smallest id in the > connected
 * component containing that vertex.
 *
 * Algorithm: propagate the smallest vertex id along the edges to all
 * vertices of a connected component.
 */
public class ConnectedComponents {
    public static class CCVertex extends
        Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
        @Override
        public void compute(
            ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context, Iterable<LongWritabl
            e> msgs) throws IOException {
            if (context.getSuperstep() == 0L) {
                this.setValue(getId());
                context.sendMessageToNeighbors(this, getValue());
                return;
            }
            long minID = Long.MAX_VALUE;
            for (LongWritable id : msgs) {
                if (id.get() < minID) { minID = id.get(); }
            }
            if (minID < this.getValue().get()) {
                this.setValue(new LongWritable(minID));
                context.sendMessageToNeighbors(this, getValue());
            } else {
                this.voteToHalt();
            }
        }
    }
    /**
     * Output Table Description:
     * +-----+-----+
     * Field | Type | Comment |
     * +-----+-----+
     * v | bigint | vertex id |
     * minID | bigint | smallest id in the connected component |
     * +-----+-----+
     */
    @Override
    public void cleanup(
        WorkerContext<LongWritable, LongWritable, NullWritable, LongWritable> context) throws IOException {
        context.write(getId(), getValue());
    }
}
/**
 * Input Table Description:

```

```

* +-----+-----+
* Field | Type | Comment |
* +-----+-----+
* v | bigint | vertex id |
* es | string | comma separated target vertex id of outgoing edges |
* +-----+-----+
*
* Example:
* For graph:
* 1 ---- 2
* | |
* 3 ---- 4
* Input table:
* +-----+
* v | es |
* +-----+
* | 1 | 2,3 |
* | 2 | 1,4 |
* | 3 | 1,4 |
* | 4 | 2,3 |
* +-----+
*/
public static class CCVertexReader extends
GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
@Override
public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context) throws IOException
{
CCVertex vertex = new CCVertex();
vertex.setId((LongWritable) record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) {
long destID = Long.parseLong(edges[i]);
vertex.addEdge(new LongWritable(destID), NullWritable.get());
}
context.addVertexRequest(vertex);
}
}

public static void main(String[] args) throws IOException {
if (args.length < 2) {
System.out.println("Usage: <input> <output>");
System.exit(-1);
}
GraphJob job = new GraphJob();
job.setGraphLoaderClass(CCVertexReader.class);
job.setVertexClass(CCVertex.class);
job.setCombinerClass(MinLongCombiner.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
long startTime = System.currentTimeMillis();
job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + " second
s");
}
}

```

9.6.7. Topological sorting

This topic provides an example of topological sorting in MaxCompute Graph.

For a directed edge (u,v) , all vertex sequences that satisfy $u < v$ are called topological sequences. Topological sorting is an algorithm that is used to calculate the topological sequence of a directed graph.

Steps to implement the algorithm:

1. Identify a vertex without incoming edges and generate an output record.
2. Delete the vertex and all its outgoing edges from the graph.
3. Repeat the preceding steps until output records are generated for all the vertices without incoming edges.

Example:

```
import java.io.IOException;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.WritableRecord;
public class TopologySort {
private final static Log LOG = LogFactory.getLog(TopologySort.class);
public static class TopologySortVertex extends
Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
@Override
public void compute(
ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context, Iterable<LongWritabl
e> messages) throws IOException {
// in superstep 0, each vertex sends message whose value is 1 to its
// neighbors
if (context.getSuperstep() == 0) { if (hasEdges()) {
context.sendMessageToNeighbors(this, new LongWritable(1L));
}
} else if (context.getSuperstep() >= 1) {
// compute each vertex's indegree
long indegree = getValue().get();
for (LongWritable msg : messages) {
indegree += msg.get();
}
setValue(new LongWritable(indegree));
if (indegree == 0) {
voteToHalt();
if (hasEdges()) {
context.sendMessageToNeighbors(this, new LongWritable(-1L));
}
}
context.write(new LongWritable(context.getSuperstep()), getId());
LOG.info("vertex: " + getId());
}
}
```

```

context.aggregate(new LongWritable(indegree));
}
}
}
public static class TopologySortVertexReader extends
GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context) throws IOException
{
TopologySortVertex vertex = new TopologySortVertex();
vertex.setId((LongWritable) record.get(0));
vertex.setValue(new LongWritable(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) {
long edge = Long.parseLong(edges[i]);
if (edge >= 0) {
vertex.addEdge(new LongWritable(Long.parseLong(edges[i])), NullWritable.get());
}
}
LOG.info(record.toString());
context.addVertexRequest(vertex);
}
}
public static class LongSumCombiner extends
Combiner<LongWritable, LongWritable> {
@Override
public void combine(LongWritable vertexId, LongWritable combinedMessage, LongWritable messageToCombine) throws IOException {
combinedMessage.set(combinedMessage.get() + messageToCombine.get());
}
}
public static class TopologySortAggregator extends
Aggregator<BooleanWritable> {
@SuppressWarnings("rawtypes")
@Override
public BooleanWritable createInitialValue(WorkerContext context) throws IOException {
return new BooleanWritable(true);
}
@Override
public void aggregate(BooleanWritable value, Object item) throws IOException {
boolean hasCycle = value.get();
boolean inDegreeNotZero = ((LongWritable) item).get() == 0 ? false : true;
value.set(hasCycle && inDegreeNotZero);
}
@Override
public void merge(BooleanWritable value, BooleanWritable partial) throws IOException {
value.set(value.get() && partial.get());
}
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, BooleanWritable value) throws IOException {
if (context.getSuperstep() == 0) {
// since the initial aggregator value is true, and in superstep we don't
// do aggregate
return false;
}
return value.get();
}
}
}

```

```
}
public static void main(String[] args) throws IOException { if (args.length != 2) {
System.out.println("Usage : <inputTable> <outputTable>");
System.exit(-1);
}
// Format of the input table:
// 0 1, 2
// 1 3
// 2 3
// 3 -1
// The first column is vertexid, and the second column is the destination vertexid of the vertex. If
the value is -1, the vertex does not have any outgoing edges.
// Format of the output table:
// 0 0
// 1 1
// 1 2
// 2 3
// The first column is the supstep value, in which the topological sequence is hidden. The second co
lumn is vertexid.
// TopologySortAggregator is used to determine whether the graph has loops.
// If the input graph has a loop and the indegree of all active vertices is not 0, the iteration end
s.
// You can use records in the input and output tables to determine whether the graph has loops.
GraphJob job = new GraphJob();
job.setGraphLoaderClass(TopologySortVertexReader.class);
job.setVertexClass(TopologySortVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
job.setCombinerClass(LongSumCombiner.class);
job.setAggregatorClass(TopologySortAggregator.class);
long startTime = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + " second
s");
}
}
```

9.6.8. Linear regression

This topic provides a linear regression example.

In statistics, linear regression is a statistical analysis method used to determine the dependency between two or more variables. Linear regression is different from the classification algorithm that predicts discrete values. The regression algorithm can predict continuous values.

The linear regression algorithm defines the loss function as the sum of the least square errors of a sample set. It solves the weight vector by minimizing the loss function.

A common solution is the gradient descent method. It is implemented in the following steps:

1. Initialize the weight vector to provide the descent speed and iterations or iteration convergence condition.
2. Calculate the least square error for each sample.
3. Calculate the sum of the least square errors and update the weight based on the descent speed.
4. Repeat iterations until convergence occurs.

Example:

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
```

```

import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
 * LinearRegression input: y,x1,x2,x3,.....
 *
 * @author shiwan.ch
 * @update jiasheng.tjs running parameters are like: tjs_lr_in > tjs_lr_out 1500 2
 * 0.07
 */
public class LinearRegression {
    public static class GradientWritable implements Writable {
        Tuple lastTheta;
        Tuple currentTheta;
        Tuple tmpGradient;
        LongWritable count;
        DoubleWritable lost;
        @Override
        public void readFields(DataInput in) throws IOException {
            lastTheta = new Tuple();
            lastTheta.readFields(in);
            currentTheta = new Tuple();
            currentTheta.readFields(in);
            tmpGradient = new Tuple();
            tmpGradient.readFields(in);
            count = new LongWritable();
            count.readFields(in);
            /* update 1: add a variable to store lost at every iteration */
            lost = new DoubleWritable();
            lost.readFields(in);
        }
        @Override
        public void write(DataOutput out) throws IOException {
            lastTheta.write(out);
            currentTheta.write(out);
            tmpGradient.write(out);
            count.write(out);
            lost.write(out);
        }
    }
    public static class LinearRegressionVertex extends
        Vertex<LongWritable, Tuple, NullWritable, NullWritable> {
        @Override
        public void compute(
            ComputeContext<LongWritable, Tuple, NullWritable, NullWritable> context, Iterable<NullWritable> mess
            ages) throws IOException {
            context.aggregate(getValue());
        }
    }
}

```

```

}
public static class LinearRegressionVertexReader extends
GraphLoader<LongWritable, Tuple, NullWritable, NullWritable> {
@Override
public void load(LongWritable recordNum, WritableRecord record, MutationContext<LongWritable, Tuple,
NullWritable, NullWritable> context)
throws IOException {
LinearRegressionVertex vertex = new LinearRegressionVertex();
vertex.setId(recordNum);
vertex.setValue(new Tuple(record.getAll())); context.addVertexRequest(vertex);
}
}
public static class LinearRegressionAggregator extends
Aggregator<GradientWritable> {
@SuppressWarnings("rawtypes")
@Override
public GradientWritable createInitialValue(WorkerContext context) throws IOException {
if (context.getSuperstep() == 0) {
/* set initial value, all 0 */
GradientWritable grad = new GradientWritable();
grad.lastTheta = new Tuple();
grad.currentTheta = new Tuple();
grad.tmpGradient = new Tuple();
grad.count = new LongWritable(1);
grad.lost = new DoubleWritable(0.0);
int n = (int) Long.parseLong(context.getConfiguration().get("Dimension"));
for (int i = 0; i < n; i++) { grad.lastTheta.append(new DoubleWritable(0));
grad.currentTheta.append(new DoubleWritable(0));
grad.tmpGradient.append(new DoubleWritable(0));
}
return grad;
} else
return (GradientWritable) context.getLastAggregatedValue(0);
}
public static double vecMul(Tuple value, Tuple theta) {
/* perform this partial computing: y(i)-hθ(x(i)) for each sample */
/* value denote a piece of sample and value(0) is y */
double sum = 0.0;
for (int j = 1; j < value.size(); j++)
sum += Double.parseDouble(value.get(j).toString()) * Double.parseDouble(theta.get(j).toString());
Double tmp = Double.parseDouble(theta.get(0).toString()) + sum -Double.parseDouble(value.get(0).toSt
ring());
return tmp;
}
@Override
public void aggregate(GradientWritable gradient, Object value) throws IOException {
/*
* perform on each vertex--each sample i: set theta(j) for each sample > i
* for each dimension
*/
double tmpVar = vecMul((Tuple) value, gradient.currentTheta);
/*
* update 2:local worker aggregate(), perform like merge() below. This
* means the variable gradient denotes the previous aggregated value
*/
gradient.tmpGradient.set(0, new DoubleWritable( ((DoubleWritable) gradient.tmpGradient.get(0)).get ()
+ tmpVar));
gradient.lost.set(Math.pow(tmpVar, 2));
/*

```

```

* calculate  $(y(i)-h\theta(x(i)))x(i)(j)$  for each sample  $i$  for each
* dimension  $j$ 
*/
for (int j = 1; j < gradient.tmpGradient.size(); j++) gradient.tmpGradient.set(j, new DoubleWritable
(
((DoubleWritable) gradient.tmpGradient.get(j)).get() + tmpVar * Double.parseDouble(((Tuple) value).g
et(j).toString())));
}
@Override
public void merge(GradientWritable gradient, GradientWritable partial) throws IOException {
/* perform SumAll on each dimension for all samples. */
Tuple master = (Tuple) gradient.tmpGradient;
Tuple part = (Tuple) partial.tmpGradient;
for (int j = 0; j < gradient.tmpGradient.size(); j++) {
DoubleWritable s = (DoubleWritable) master.get(j);
s.set(s.get() + ((DoubleWritable) part.get(j)).get());
}
gradient.lost.set(gradient.lost.get() + partial.lost.get());
}
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, GradientWritable gradient) throws IOException {
/*
* 1. calculate new theta 2. judge the diff between last step and this
* step, if smaller than the threshold, stop iteration
*/
gradient.lost = new DoubleWritable(gradient.lost.get() / (2 * context.getTotalNumVertices()));
/*
* we can calculate lost in order to make sure the algorithm is running > on
* the right direction (for debug)
*/
System.out.println(gradient.count + " lost:" + gradient.lost);
Tuple tmpGradient = gradient.tmpGradient;
System.out.println("tmpGra" + tmpGradient);
Tuple lastTheta = gradient.lastTheta;
Tuple tmpCurrentTheta = new Tuple(gradient.currentTheta.size());
System.out.println(gradient.count + " terminate_start_last:" + lastTheta);
double alpha = 0.07; // learning rate
// alpha =
// Double.parseDouble(context.getConfiguration().get("Alpha"));
/* perform  $\theta(j) = \theta(j) - \alpha * \text{tmpGradient}$  */
long M = context.getTotalNumVertices();
/*
* update 3: add (/M) on the code. The original code forget this step
*/
for (int j = 0; j < lastTheta.size(); j++) { tmpCurrentTheta
.set(j,
new DoubleWritable(Double.parseDouble(lastTheta.get(j)
.toString()) - alpha / M * Double.parseDouble(tmpGradient.get(j).toString())));
}
System.out.println(gradient.count + " terminate_start_current:" + tmpCurrentTheta);
// judge if convergence is happening.
double diff = 0.00d;
for (int j = 0; j < gradient.currentTheta.size(); j++)
diff += Math.pow(((DoubleWritable) tmpCurrentTheta.get(j)).get() - ((DoubleWritable) lastTheta.get(j)
).get(), 2);
if (
/*
*  $\text{Math.sqrt(diff)} < 0.00000000005d$  ||

```

```

*/
Long.parseLong(context.getConfiguration().get("Max_Iter_Num")) == gradient.count
.get()) { context.write(gradient.currentTheta.toArray());
return true;
}
gradient.lastTheta = tmpCurrentTheta;
gradient.currentTheta = tmpCurrentTheta;
gradient.count.set(gradient.count.get() + 1);
int n = (int) Long.parseLong(context.getConfiguration().get("Dimension"));
/*
* update 4: Important!!! Remember this step. Graph won't reset the
* initial value for global variables at the beginning of each iteration
*/
for (int i = 0; i < n; i++) {
gradient.tmpGradient.set(i, new DoubleWritable(0));
}
return false;
}
}

public static void main(String[] args) throws IOException { GraphJob job = new GraphJob();
job.setGraphLoaderClass(LinearRegressionVertexReader.class); job.setRuntimePartitioning(false);
job.setNumWorkers(3);
job.setVertexClass(LinearRegressionVertex.class);
job.setAggregatorClass(LinearRegressionAggregator.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
job.setMaxIteration(Integer.parseInt(args[2])); // Numbers of Iteration
job.setInt("Max_Iter_Num", Integer.parseInt(args[2]));
job.setInt("Dimension", Integer.parseInt(args[3])); // Dimension
job.setFloat("Alpha", Float.parseFloat(args[4])); // Learning rate
long start = System.currentTimeMillis(); job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}
}

```

9.6.9. Triangle count

This topic provides a triangle count example.

The triangle count algorithm calculates the number of triangles that pass through each vertex in a graph. The algorithm is implemented in the following steps:

1. Send the ID of each vertex to all its outgoing neighbors.
2. Store information about incoming and outgoing neighbors, and send the information to outgoing neighbors.
3. Calculate the number of endpoint intersections for each edge, calculate the sum, and write the output results to a table.
4. Sum up the output results in the table and divide the sum by 3 to obtain the number of triangles that pass through each vertex.

Example:

```

import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;

```

```

import com.aliyun.odps.graph.ReduceContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
 * Compute the number of triangles passing through each vertex.
 *
 * The algorithm can be computed in three supersteps:
 * I. Each vertex sends a message with its ID to all its outgoing
 * neighbors.
 * II. The incoming neighbors and outgoing neighbors are stored and
 * send to outgoing neighbors.
 * III. For each edge compute the intersection of the sets at destination
 * vertex and sum them, then output to table.
 *
 * The triangle count is the sum of output table and divide by three > since
 * each triangle is counted three times.
 */
public class TriangleCount {
    public static class TCVertex extends
    Vertex<LongWritable, Tuple, NullWritable, Tuple> {
        @Override
        public void setup(
        WorkerContext<LongWritable, Tuple, NullWritable, Tuple> context) throws IOException {
            // collect the outgoing neighbors
            Tuple t = new Tuple();
            if (this.hasEdges()) {
                for (Edge<LongWritable, NullWritable> edge : this.getEdges()) {
                    t.append(edge.getDestVertexId());
                }
            }
            this.setValue(t);
        }
        @Override
        public void compute(
        ComputeContext<LongWritable, Tuple, NullWritable, Tuple> context, Iterable<Tuple> msgs) throws IOException {
            if (context.getSuperstep() == 0L) {
                // sends a message with its ID to all its outgoing neighbors
                Tuple t = new Tuple(); t.append(getId());
                context.sendMessageToNeighbors(this, t);
            } else if (context.getSuperstep() == 1L) {
                // store the incoming neighbors
                for (Tuple msg : msgs) {
                    for (Writable item : msg.getAll()) {
                        if (! this.getValue().getAll().contains((LongWritable)item)) {
                            this.getValue().append((LongWritable)item);
                        }
                    }
                }
            }
            // send both incoming and outgoing neighbors to all outgoing neighbors
            context.sendMessageToNeighbors(this, getValue());
        } else if (context.getSuperstep() == 2L) {
            // count the sum of intersection at each edge
            long count = 0;

```

```

long count = 0;
for (Tuple msg : msgs) {
    for (Writable id : msg.getAll()) {
        if (getValue().getAll().contains(id)) { count ++;
        }
    }
}
// output to table
context.write(getId(), new LongWritable(count));
this.voteToHalt();
}
}
}

public static class TCVertexReader extends
GraphLoader<LongWritable, Tuple, NullWritable, Tuple> {
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, Tuple, NullWritable, Tuple> context) throws IOException {
TCVertex vertex = new TCVertex();
vertex.setId((LongWritable) record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) { try {
long destID = Long.parseLong(edges[i]);
vertex.addEdge(new LongWritable(destID), NullWritable.get());
} catch (NumberFormatException nfe) { System.err.println("Ignore " + nfe);
}
}
context.addVertexRequest(vertex);
}
}

public static void main(String[] args) throws IOException { if (args.length < 2) {
System.out.println("Usage: <input> <output>"); System.exit(-1);
}
GraphJob job = new GraphJob();
job.setGraphLoaderClass(TCVertexReader.class);
job.setVertexClass(TCVertex.class);
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
long startTime = System.currentTimeMillis();
job.run();
System.out.println("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + " second
s");
}
}
}

```

9.6.10. Edge table import

This topic provides an example of importing an edge table.

Example:

```

import java.io.IOException;
import com.aliyun.odps.conf.Configuration;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.VertexResolver;

```

```

import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableComparable;
import com.aliyun.odps.io.WritableRecord;
/**
 * This example describes how to compile a Graph job program to load data of different data types. It
 * covers how GraphLoader
 * and VertexResolver are used together to build a graph.
 *
 * A MaxCompute Graph job uses MaxCompute tables as the input. For example, a job uses two tables as
 * the input. One stores information about vertices, and the other stores information about edges.
 * Format of the table that stores information about vertices:
 * +-----+
 * | VertexID | VertexValue |
 * +-----+
 * |      id0|          9|
 * +-----+
 * |      id1|          7|
 * +-----+
 * |      id2|          8|
 * +-----+
 *
 * Format of the table that stores information about edges:
 * +-----+
 * | VertexID | DestVertexID| EdgeValue|
 * +-----+
 * |      id0|          id1|          1|
 * +-----+
 * |      id0|          id2|          2|
 * +-----+
 * |      id2|          id1|          3|
 * +-----+
 *
 * The two tables show that id0 has two outgoing edges that point to id1 and id2. id2 has one outgoing
 * edge that points to id1, and id1 has no outgoing edges.
 *
 * For data of this type, in GraphLoader::load(LongWritable, Record, MutationContext), MutationContext#addVertexRequest(Vertex) can be used to add vertices to the graph.
 * link MutationContext#addEdgeRequest(WritableComparable, Edge) can be used to add edges to the graph. In
 * link VertexResolver#resolve(WritableComparable, Vertex, VertexChanges, boolean),
 * vertices and edges added by using the load() method are combined to a vertex object. This object
 * is used as the return value and added to the graph for computing.
 */
public class VertexInputFormat {
    private final static String EDGE_TABLE = "edge.table";
    /**
     * Resolve records to vertices and edges. Each record indicates a vertex or edge based on its source.
     *
     * <p>
     * The following process is similar to the map stage of com.aliyun.odps.mapreduce.Mapper.
     * Enter a record to generate key-value pairs.
     * The keys are vertex IDs, and the values are vertices or edges that are written based on context.
     * These key-value pairs are aggregated by using LoadingVertexResolver based on the vertex IDs.
     *
     * Note: The vertices or edges added here are requests sent based on the record content and are no

```

```

t used in computing.
 * Only the vertices or edges added by using VertexResolver are used for computing.
 **/
public static class VertexInputLoader extends
    GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
    private boolean isEdgeData;
    /**
     * Configure VertexInputLoader.
     *
     * @param conf
     *         Indicates the configured parameters of a job. These parameters are configured in the
     main function of GraphJob or by running the SET command on the client.
     * @param workerId
     *         Indicates the serial number of the running worker. The number starts from 0 and can
     be used to build a unique vertex ID.
     * @param inputTableInfo
     *         Indicates the information about the input table that is loaded to the running worker
     . The information can be used to determine the data type of the input data (the format of the record
     ).
     **/
    @Override
    public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
        isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableInfo());
    }
    /**
     * Resolve the record to edges based on the record content and send a request to add them to the
     graph.
     *
     * @param recordNum
     *         Indicates the serial number of the record. The number starts from 1 and is separatel
     y counted in each worker.
     * @param record
     *         Indicates the records in the input table. The table contains three columns, which in
     dicate the source vertex, destination vertex, and edge weight.
     * @param context
     *         Indicates the context. The context is used when you send a request to add resolved e
     dges to the graph.
     **/
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
        throws IOException {
        if (isEdgeData) {
            /**
             * Data comes from the table that stores information about edges.
             *
             * 1. The first column indicates the IDs of source vertices.
             **/
            LongWritable sourceVertexID = (LongWritable) record.get(0);
            /**
             * 2. The second column indicates the IDs of destination vertices.
             **/
            LongWritable destinationVertexID = (LongWritable) record.get(1);
            /**
             * 3. The third column indicates edge weights.
             **/
            LongWritable edgeValue = (LongWritable) record.get(2);

```

```

/**
 * 4. Create an edge based on a destination vertex ID and an edge weight.
 */
Edge<LongWritable, LongWritable> edge = new Edge<LongWritable, LongWritable>(
    destinationVertexID, edgeValue);
/**
 * 5. Send a request to add the edge to a source vertex.
 */
context.addEdgeRequest(sourceVertexID, edge);
/**
 * 6. If each record indicates a bidirectional edge, repeat Step 4 and Step 5.
 * Edge<LongWritable, LongWritable> edge2 = new
 * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
 * context.addEdgeRequest(destinationVertexID, edge2);
 */
} else {
/**
 * Data comes from the table that stores information about vertices.
 *
 * 1. The first column indicates the IDs of vertices.
 */
LongWritable vertexID = (LongWritable) record.get(0);
/**
 * 2. The second column indicates the values of vertices.
 */
LongWritable vertexValue = (LongWritable) record.get(1);
/**
 * 3. Create a vertex based on an ID and a value.
 */
MyVertex vertex = new MyVertex();
/**
 * 4. Initialize the vertex.
 */
vertex.setId(vertexID);
vertex.setValue(vertexValue);
/**
 * 5. Send a request to add the vertex.
 */
context.addVertexRequest(vertex);
}
}
/**
 * Summarize key-value pairs generated by using GraphLoader::load(LongWritable, Record, MutationContext).
 * This process is similar to the reduce stage of com.aliyun.odps.mapreduce.Reducer. For a unique
 * vertex ID, all actions, such as
 * adding or removing vertices or edges for the ID, are stored in VertexChanges.
 *
 * Note: Not only conflicting vertices or edges added by using the load() method are called. A conflict
 * occurs when multiple same vertex objects or duplicate edges are added.
 * All the IDs that are requested to be generated by using the load() method are called.
 */
public static class LoadingResolver extends
    VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
/**
 * Process a request to add or remove vertices or edges for an ID.
 *
 * VertexChanges has four APIs, which correspond to the four APIs of MutationContext:

```

```

    * VertexChanges::getAddedVertexList() corresponds to
    * MutationContext::addVertexRequest(Vertex).
    * In the load() method, if a request is sent to add vertex objects with the same ID, the objects
    s are collected to the returned list.
    * VertexChanges::getAddedEdgeList() corresponds to
    * MutationContext::addEdgeRequest(WritableComparable, Edge).
    * If a request is sent to add edge objects with the same source vertex ID, the objects are coll
    ected to the returned list.
    * VertexChanges::getRemovedVertexCount() corresponds to
    * MutationContext::removeVertexRequest(WritableComparable).
    * If a request is sent to remove vertices with the same ID, the number of total removal request
    s is returned.
    * VertexChanges#getRemovedEdgeList() corresponds to
    * MutationContext#removeEdgeRequest(WritableComparable, WritableComparable).
    * If a request is sent to remove edge objects with the same source vertex ID, the objects are c
    ollected to the returned list.
    *
    * You can process the changes in the ID and state whether the ID is used in computing in the re
    turn value.
    * If the returned vertex is not null, the ID is used in subsequent computing. If the returned v
    ertex is null, the ID is not used in subsequent computing.
    *
    * @param vertexId
    *         Indicates the ID of the vertex that is requested to be added or the source vertex ID
    of the edge that is requested to be added.
    * @param vertex
    *         Indicates an existing vertex object. The value of this parameter is always null in t
    he data loading phase.
    * @param vertexChanges
    *         Indicates a collection of vertices or edges that are requested to be added or remove
    d for the ID.
    * @param hasMessages
    *         Indicates whether messages are sent to the ID. The value of this parameter is always
    false in the data loading phase.
    */
    @Override
    public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve(
        LongWritable vertexId,
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
        VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable> vertexChanges,
        boolean hasMessages) throws IOException {
    /**
    * 1. Obtain the vertex object for computing.
    */
    MyVertex computeVertex = null;
    if (vertexChanges.getAddedVertexList() == null
        || vertexChanges.getAddedVertexList().isEmpty()) {
        computeVertex = new MyVertex();
        computeVertex.setId(vertexId);
    } else {
    /**
    * Each record indicates a unique vertex in the table that stores information about vertices
    .
    */
        computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);
    }
    /**
    * 2. Add the edge, which is requested to be added to the vertex, to the vertex object. If the
    data is a possible duplicate, deduplicate it based on algorithm requirements.

```

```

    /**
    if (vertexChanges.getAddedEdgeList() != null) {
        for (Edge<LongWritable, LongWritable> edge : vertexChanges
            .getAddedEdgeList()) {
            computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
        }
    }
    /**
    * 3. Return the vertex object and add it to the final graph for computing.
    */
    return computeVertex;
}
}
/**
* Determine the actions of the vertex that is used in computing.
*
*/
public static class MyVertex extends
    Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
    /**
    * Write the edge of the vertex to the result table based on the format of the input table. Make
    sure that the formats and data of the input and output tables are the same.
    *
    * @param context
    *     Indicates the runtime context.
    * @param messages
    *     Indicates the input messages.
    */
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
        Iterable<LongWritable> messages) throws IOException {
    /**
    * Write the ID and value of the vertex to the result table that stores information about vert
    ices.
    */
    context.write("vertex", getId(), getValue());
    /**
    * Write the edge of the vertex to the result table that stores information about edges.
    */
    if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
            context.write("edge", getId(), edge.getDestVertexId(),
                edge.getValue());
        }
    }
    /**
    * Perform only one iteration.
    */
    voteToHalt();
}
}
/**
* @param args
* @throws IOException
*/
public static void main(String[] args) throws IOException {
    if (args.length < 4) {
        throw new IOException(

```

```

        "Usage: VertexInputFormat <vertex input> <edge input> <vertex output> <edge output>");
    }
    /**
     * Use GraphJob to configure a Graph job.
     */
    GraphJob job = new GraphJob();
    /**
     * 1. Specify input graph data and the table that stores information about edges.
     */
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addInput(TableInfo.builder().tableName(args[1]).build());
    job.set(EDGE_TABLE, args[1]);
    /**
     * 2. Specify the data loading mode and resolve the records to edges. This process is similar to
     the map stage. The generated key is the vertex ID, and the generated value is the edge.
     */
    job.setGraphLoaderClass(VertexInputLoader.class);
    /**
     * 3. Specify the data loading phase to generate the vertex that is used in computing. This proc
     ess is similar to the reduce stage. In the reduce stage, edges that are generated in the map stage a
     re combined into a vertex.
     */
    job.setLoadingVertexResolverClass>LoadingResolver.class);
    /**
     * 4. Specify the actions of the vertex that is used in computing. The vertex.compute() method i
     s used for each iteration.
     */
    job.setVertexClass(MyVertex.class);
    /**
     * 5. Specify the result table of the Graph job and write the computing results to the result ta
     ble.
     */
    job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
    job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());
    /**
     * 6. Submit the job for execution.
     */
    job.run();
    }
}

```

9.6.11. Vertex table import

This topic provides an example of importing a vertex table.

Example:

```

import java.io.IOException;
import com.aliyun.odps.conf.Configuration;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.VertexResolver;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.graph.Edge;

```

```

import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableComparable;
import com.aliyun.odps.io.WritableRecord;
/**
 * This example describes how to compile a Graph job program to load data of different data types. It
 * covers how GraphLoader and VertexResolver are used together to build a graph.
 * A MaxCompute Graph job uses MaxCompute tables as the input. For example, a job uses two tables as
 * the input. One stores information about vertices, and the other stores information about edges.
 * Format of the table that stores information about vertices:
 * +-----+
 * | VertexID | VertexValue |
 * +-----+
 * |      id0|         9|
 * +-----+
 * |      id1|         7|
 * +-----+
 * |      id2|         8|
 * +-----+
 *
 * Format of the table that stores information about edges:
 * +-----+
 * | VertexID | DestVertexID| EdgeValue|
 * +-----+
 * |      id0|         id1|         1|
 * +-----+
 * |      id0|         id2|         2|
 * +-----+
 * |      id2|         id1|         3|
 * +-----+
 *
 * The two tables show that id0 has two outgoing edges that point to id1 and id2. id2 has one outgoing
 * edge that points to id1, and id1 has no outgoing edges.
 *
 * For data of this type, in GraphLoader::load(LongWritable, Record, MutationContext),
 * MutationContext#addVertexRequest(Vertex) can be used to add vertices to the graph.
 * link MutationContext#addEdgeRequest(WritableComparable, Edge) can be used to add edges to the gra
 * ph. In
 * link VertexResolver#resolve(WritableComparable, Vertex, VertexChanges, boolean)
 * vertices and edges added by using the load() method are combined to a vertex object. This object
 * is used as the return value and added to the graph for computing.
 */
**/
public class VertexInputFormat {
    private final static String EDGE_TABLE = "edge.table";
    /**
     * Resolve records to vertices and edges. Each record indicates a vertex or edge based on its sour
     * ce.
     *
     * The following process is similar to the map stage of com.aliyun.odps.mapreduce.Mapper. Enter a
     * record to generate key-value pairs.
     * The keys are vertex IDs, and the values are vertices or edges that are written based on context
     * . These key-value pairs are aggregated by using LoadingVertexResolver based on the vertex IDs.
     *
     * Note: The vertices or edges added here are requests sent based on the record content and are no
     * t used in computing.
     * Only the vertices or edges added by using VertexResolver are used for computing.
     */
    **/
    public static class VertexInputLoader extends
        GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {

```

```

private boolean isEdgeData;
/**
 * Configure VertexInputLoader.
 *
 * @param conf
 *      Indicates the configured parameters of a job. These parameters are configured in the
Main function of GraphJob or by running the SET command in the console.
 * @param workerId
 *      Indicates the serial number of the running worker. The number starts from 0 and can
be used to build a unique vertex ID.
 * @param inputTableInfo
 *      Indicates the information about the input table that is loaded to the running worker
. The information can be used to determine the data type of the input data (the format of the record
).
 */
@Override
public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
    isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableInfo());
}
/**
 * Resolve the record to edges based on the record content and send a request to add them to the
graph.
 *
 * @param recordNum
 *      Indicates the serial number of the record. The number starts from 1 and is separatel
y counted in each worker.
 * @param record
 *      Indicates the records in the input table. The table contains three columns, which in
dicate the source vertex, destination vertex, and edge weight.
 * @param context
 *      Indicates the context. The context is used when you send a request to add resolved e
dges to the graph.
 */
@Override
public void load(
    LongWritable recordNum,
    WritableRecord record,
    MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
    throws IOException {
    if (isEdgeData) {
        /**
         * Data comes from the table that stores information about edges.
         *
         * 1. The first column indicates the IDs of source vertices.
         */
        LongWritable sourceVertexID = (LongWritable) record.get(0);
        /**
         * 2. The second column indicates the IDs of destination vertices.
         */
        LongWritable destinationVertexID = (LongWritable) record.get(1);
        /**
         * 3. The third column indicates edge weights.
         */
        LongWritable edgeValue = (LongWritable) record.get(2);
        /**
         * 4. Create an edge based on a destination vertex ID and an edge weight.
         */
        Edge<LongWritable, LongWritable> edge = new Edge<LongWritable, LongWritable>(
            destinationVertexID, edgeValue);
    }
}

```

```

/**
 * 5. Send a request to add the edge to a source vertex.
 */
context.addEdgeRequest(sourceVertexID, edge);
/**
 * 6. If each record indicates a bidirectional edge, repeat Step 4 and Step 5.
 * Edge<LongWritable, LongWritable> edge2 = new
 * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
 * context.addEdgeRequest(destinationVertexID, edge2);
 */
} else {
/**
 * Data comes from the table that stores information about vertices.
 *
 * 1. The first column indicates the IDs of vertices.
 */
LongWritable vertexID = (LongWritable) record.get(0);
/**
 * 2. The second column indicates the values of vertices.
 */
LongWritable vertexValue = (LongWritable) record.get(1);
/**
 * 3. Create a vertex based on an ID and a value.
 */
MyVertex vertex = new MyVertex();
/**
 * 4. Initialize the vertex.
 */
vertex.setId(vertexID);
vertex.setValue(vertexValue);
/**
 * 5. Send a request to add the vertex.
 */
context.addVertexRequest(vertex);
}
}
/**
 * Summarize key-value pairs generated by using GraphLoader::load(LongWritable, Record, MutationContext).
 * This process is similar to the reduce stage of com.aliyun.odps.mapreduce.Reducer. For a unique
 * vertex ID, all actions, such as
 * adding or removing vertices or edges for the ID, are stored in VertexChanges.
 *
 * Note: Not only conflicting vertices or edges added by using the load() method are called. A conflict
 * occurs when multiple same vertex objects or duplicate edges are added.
 * All the IDs that are requested to be generated by using the load() method are called.
 */
public static class LoadingResolver extends
    VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
/**
 * Process a request to add or remove vertices or edges for an ID.
 *
 * VertexChanges has four APIs, which correspond to the four APIs of MutationContext:
 * VertexChanges::getAddedVertexList() corresponds to
 * MutationContext::addVertexRequest(Vertex).
 * In the load() method, if a request is sent to add vertex objects with the same ID, the objects
 * are collected to the returned list.
 * VertexChanges::getAddedEdgeList() corresponds to

```

```

    * MutationContext::addEdgeRequest(WritableComparable, Edge)
    * If a request is sent to add edge objects with the same source vertex ID, the objects are collected to the returned list.
    * VertexChanges::getRemovedVertexCount() corresponds to
    * MutationContext::removeVertexRequest(WritableComparable)
    * If a request is sent to remove vertices with the same ID, the number of total removal requests is returned.
    * VertexChanges#getRemovedEdgeList() corresponds to
    * MutationContext#removeEdgeRequest(WritableComparable, WritableComparable)
    * If a request is sent to remove edge objects with the same source vertex ID, the objects are collected to the returned list.
    *
    * You can process the changes in the ID and state whether the ID is used in computing in the return value.
    * If the returned vertex is not null, the ID is used in subsequent computing. If the returned vertex is null, the ID is not used in subsequent computing.
    *
    * @param vertexId
    *     Indicates the ID of the vertex that is requested to be added or the source vertex ID of the edge that is requested to be added.
    * @param vertex
    *     Indicates an existing vertex object. The value of this parameter is always null in the data loading phase.
    * @param vertexChanges
    *     Indicates a collection of vertices or edges that are requested to be added or removed for the ID.
    * @param hasMessages
    *     Indicates whether messages are sent to the ID. The value of this parameter is always false in the data loading phase.
    */
    @Override
    public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve(
        LongWritable vertexId,
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
        VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable> vertexChanges,
        boolean hasMessages) throws IOException {
    /**
    * 1. Obtain the vertex object for computing.
    */
    MyVertex computeVertex = null;
    if (vertexChanges.getAddedVertexList() == null
        || vertexChanges.getAddedVertexList().isEmpty()) {
        computeVertex = new MyVertex();
        computeVertex.setId(vertexId);
    } else {
    /**
    * Each record indicates a unique vertex in the table that stores information about vertices
    *
    */
        computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);
    }
    /**
    * 2. Add the edge, which is requested to be added to the vertex, to the vertex object. If the data is a possible duplicate, deduplicate it based on algorithm requirements.
    */
    if (vertexChanges.getAddedEdgeList() != null) {
        for (Edge<LongWritable, LongWritable> edge : vertexChanges
            .getAddedEdgeList()) {
            computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
        }
    }
}

```

```

    }
}
/**
 * 3. Return the vertex object and add it to the final graph for computing.
 */
return computeVertex;
}
}
/**
 * Determine the actions of the vertex that is used in computing.
 *
 */
public static class MyVertex extends
    Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
/**
 * Write the edge of the vertex to the result table based on the format of the input table. Make
sure that the formats and data of the input and output tables are the same.
 *
 * @param context
 *     Indicates the runtime context.
 * @param messages
 *     Indicates the input messages.
 */
@Override
public void compute(
    ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
    Iterable<LongWritable> messages) throws IOException {
/**
 * Write the ID and value of the vertex to the result table that stores information about vert
ices.
 */
context.write("vertex", getId(), getValue());
/**
 * Write the edge of the vertex to the result table that stores information about edges.
 */
if (hasEdges()) {
    for (Edge<LongWritable, LongWritable> edge : getEdges()) {
        context.write("edge", getId(), edge.getDestVertexId(),
            edge.getValue());
    }
}
/**
 * Perform only one iteration.
 */
voteToHalt();
}
}
/**
 * @param args
 * @throws IOException
 */
public static void main(String[] args) throws IOException {
    if (args.length < 4) {
        throw new IOException(
            "Usage: VertexInputFormat <vertex input> <edge input> <vertex output> <edge output>");
    }
/**
 * Use GraphJob to configure a Graph job.
 */
}
}

```

```
GraphJob job = new GraphJob();
/**
 * 1. Specify input graph data and the table that stores information about edges.
 */
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addInput(TableInfo.builder().tableName(args[1]).build());
job.set(EDGE_TABLE, args[1]);
/**
 * 2. Specify the data loading mode and resolve the records to edges. This process is similar to
the map stage. The generated key is the vertex ID, and the generated value is the edge.
 */
job.setGraphLoaderClass(VertexInputLoader.class);
/**
 * 3. Specify the data loading phase to generate the vertex that is used in computing. This proc
ess is similar to the reduce stage. In the reduce stage, edges that are generated in the map stage a
re combined into a vertex.
 */
job.setLoadingVertexResolverClass>LoadingResolver.class);
/**
 * 4. Specify the actions of the vertex that is used in computing. The vertex.compute() method i
s used for each iteration.
 */
job.setVertexClass(MyVertex.class);
/**
 * 5. Specify the result table of the Graph job and write the computing results to the result ta
ble.
 */
job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());
/**
 * 6. Submit the job for execution.
 */
job.run();
}
}
```

10. Java SDK

This topic describes MaxCompute SDK for Java.

MaxCompute SDK for Java provides a variety of APIs to support development on MaxCompute.

For more information about the APIs, see *MaxCompute Developer Guide*.

11. PyODPS

11.1. Getting started

PyODPS is MaxCompute SDK for Python. PyODPS provides easy-to-use Python programming interfaces. Similar to pandas, PyODPS provides fast, flexible, and expressive data structures. The data processing feature of PyODPS is similar to that of pandas. You can use the data processing feature of PyODPS by calling the DataFrame API provided by PyODPS. This topic describes how to use PyODPS in your projects.

Background information

You can develop most programs in MaxCompute by using SQL statements. However, you must use Python to develop complex business logic and user defined functions (UDFs). For example, you must use Python in the following scenarios:

- **Interface connection:** An external service must provide required information to complete authentication before the service can access a data record in MaxCompute tables by using HTTP interfaces.
- **Asynchronous invocation:** In most cases, the system creates a node for each of thousands of tasks that have similar data processing logic. These nodes are difficult to manage and occupy excessive resources at the same time. PyODPS allows you to use queues to asynchronously run SQL tasks at a high concurrency. Queues also help you manage all nodes in a centralized manner.
- **UDF development:** If the built-in functions of MaxCompute cannot meet your business requirements, you can develop UDFs. For example, you can develop a UDF to format data of the DECIMAL type with the thousands separator.
- **SQL performance improvement:** To check whether database transactions follow the first in, first out (FIFO) rule, you must compare each new record with historical records in the only transaction table. In most cases, this transaction table contains a large number of records. If you use SQL statements to complete the comparison, the time complexity is $O(N^2)$. In addition, the SQL statements may fail to return the expected result. To resolve this issue, you can use the cache method in Python code to traverse records in the transaction table only once. In this case, the time complexity is $O(N)$, and the efficiency is significantly improved.
- **Other scenarios:** You can use Python to develop an amount allocation model. For example, you can develop a model to allocate USD 10 to three users. In this model, you must define the logic for handling the cash over and short.

Procedure

This topic describes how to use a PyODPS node in DataWorks for development.

1. Log on to the DataWorks console.
2. Create a PyODPS 2 node.
3. Configure and run the PyODPS 2 node.

i. Write the code of the PyODPS 2 node.

Write the test code in the code editor of the PyODPS 2 node. In this example, write the following code in the code editor. This example covers a full range of table operations.

```
from odps import ODPS
import sys
reload(sys)
# Change the default encoding format to UTF-8. You must execute this statement if the data c
ontains Chinese characters.
sys.setdefaultencoding('utf8')
# Create a non-partitioned table named my_new_table. The non-partitioned table contains the
fields that are of the specified data types and have the specified names.
table = o.create_table('my_new_table', 'num bigint, id string', if_not_exists=True)
# Write data to the my_new_table table.
records = [[111, 'aaa'],
           [222, 'bbb'],
           [333, 'ccc'],
           [444, 'Chinese']]
o.write_table(table, records)
# Read data from the my_new_table table.
for record in o.read_table(table):
    print record[0],record[1]
# Execute an SQL statement to read data from the my_new_table table.
result = o.execute_sql('select * from my_new_table;',hints={'odps.sql.allow.fullscan': 'true
'})
# Obtain the execution result of the SQL statement.
with result.open_reader() as reader:
    for record in reader:
        print record[0],record[1]
# Delete the table.
table.drop()
```

ii. Run the code.

After you write the code, click the  icon on the toolbar. After the code is run, you can view the result of the PyODPS 2 node on the Run Log tab.

11.2. Installation guide

This topic describes how to install PyODPS.

If you can access the Internet, we recommend that you use the Python package installer pip to install PyODPS. For more information, see [pip installation](#). If you want to speed up the download, we recommend that you use [Alibaba Cloud images](#).

Prerequisites

The following requirements are met before you install PyODPS:

- The setuptools version is 3.0 or later.
- The requests version is 2.4.0 or later.

Installation commands for reference:

```
pip install setuptools>=3.0
pip install requests>=2.4.0
```

Installation suggestions

We recommend that you install the following tools to accelerate Tunnel-based data upload:

- Greenlet. Recommended version: 0.4.10 or later.
- Cython. Recommended version: 0.19.0 or later.

The following installation commands are for your reference:

```
pip install greenlet>=0.4.10 # Optional. It accelerates Tunnel-based data upload.
pip install cython>=0.19.0 # Optional. We recommend that you do not install Cython if you use a Windows operating system.
```

 **Note** If you use a Windows operating system, make sure that you have installed Visual C++ and Cython of correct versions. Otherwise, you cannot accelerate Tunnel-based data upload. For more information about the versions of Visual C++ and Cython, see [WindowsCompilers](#).

Installation procedure

1. Run the following command to install PyODPS:

```
pip install pyodps
```

2. Run the following command to check whether the installation is successful:

```
python -c "from odps import ODPS"
```

3. If the Python version is not the default version, run the following command to switch to the default version after you have installed pip:

```
/home/tops/bin/python2.7 -m pip install setuptools>=3.0
```

11.3. Platform instructions

11.3.1. Overview

PyODPS can be called as a data development node on a data development platform such as DataWorks. The platform provides a PyODPS running environment and can schedule and run nodes. You do not need to manually create a MaxCompute entry object.

To migrate PyODPS nodes from a data development platform to a manually deployed PyODPS environment, read the instructions in the following topics:

11.3.2. Use local PyODPS

If you want to debug PyODPS locally or the resources on the platform where PyODPS is deployed cannot meet your requirements, you can deploy a local PyODPS environment. This topic describes how to deploy a local PyODPS environment.

1. Install PyODPS in a local environment. For more information, see [Installation instructions](#).
2. Create a MaxCompute entry object in the local environment.

You can execute the following statement on the data development platform to generate a template of the statement for creating a MaxCompute entry object. Then, you can modify the template to obtain the required statement.

```
print("\nfrom odps import ODPS\nno = ODPS(%r, '<access-key>', %r, '<endpoint>')\n" % (o.account.access_id, o.project))
```

3. Place the obtained statement at the beginning of all code.

11.3.3. Use PyODPS in DataWorks

This topic describes how to use PyODPS in DataWorks and the limits.

Create a PyODPS node

You can create a PyODPS node for a workflow.

 **Note** For more information, see [Create a PyODPS node](#).

MaxCompute entry

The PyODPS node in DataWorks contains a global variable `odps` or `o`, which is the MaxCompute entry. You do not need to manually define the MaxCompute entry.

```
print(o.exist_table('pyodps_iris'))
```

Execute SQL statements

You can execute SQL statements in the PyODPS node. For more information, see [SQL](#).

By default, `InstanceTunnel` is disabled in DataWorks, and `instance.open_reader` is executed by using the Result interface. In this case, a maximum of 10,000 data records can be read. You can use `reader.count` to obtain the number of data records. If you need to obtain all data iteratively, you must disable the limit on the data volume. You can execute the following statements to enable `InstanceTunnel` and disable the limit.

```
options.tunnel.use_instance_tunnel = True
options.tunnel.limit_instance_tunnel = False # Disable the limit on the data volume.
with instance.open_reader() as reader:
    # Use InstanceTunnel to read all data.
```

You can add `tunnel=True` to `open_reader` to enable `InstanceTunnel` for the current `open_reader` operation.

You can add `limit=False` to `open_reader` to disable the limit on the data volume for the current `open_reader` operation.

```
with instance.open_reader(tunnel=True, limit=False) as reader:
    # The current open_reader operation is executed by using InstanceTunnel, and all data can be read.
```

 **Note** If you do not enable `InstanceTunnel`, the format of the obtained data may be invalid.

DataFrame

- Perform operations on DataFrames.

To perform operations on DataFrames in DataWorks, you must explicitly call automatically executed methods, such as `execute` and `head`.

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
for record in iris[iris.sepal_width < 3].execute(): # Call an automatically executed method to process each data record.
```

To call an automatically executed method for data display, set `options.interactive` to `True`.

```

from odps import options
from odps.df import DataFrame
options.interactive = True # Set options.interactive to True at the beginning of the code.
iris = DataFrame(o.get_table('pyodps_iris'))
print(iris.sepal_width.sum()) # The method is executed immediately when the system displays information.

```

- Display details.

To display details, you must set `options.verbose` to True. By default, this parameter is set to True in DataWorks. The system displays details such as the Logview URL during the running process.

Obtain scheduling parameters

Different from SQL nodes in DataWorks, a PyODPS node does not replace strings such as `${param_name}` in the code. Instead, it adds a dictionary named `args` as a global variable before it runs the code. You can obtain the scheduling parameters from the dictionary. This way, the Python code is not affected. For example, on the **Scheduling configuration** tab of a PyODPS node in DataWorks, you can specify `ds=${yyyymmdd}` in the **Parameters** field in the Basic properties section. Then, you can specify the following commands in the code of the node to obtain the parameter value:

```

print('ds=' + args['ds'])
ds=yyyymmdd

```

Note You can run the following command to obtain the partition named `ds=${yyyymmdd}`:

```
o.get_table('table_name').get_partition('ds=' + args['ds'])
```

Limits on functions

- The Python version of a PyODPS node is 2.7.
- Each PyODPS node can process a maximum of 50 MB data and can occupy a maximum of 1 GB memory. Otherwise, DataWorks terminates the PyODPS node. Do not write unnecessary Python data processing code in PyODPS nodes.
- Writing and debugging code in DataWorks is inefficient. We recommend that you install an IDE locally to write code.
- To avoid excess pressure on the gateway of DataWorks, DataWorks limits the CPU utilization and memory usage. If the system displays Got killed, the memory usage exceeds the limit, and the system terminates the related processes. Therefore, we recommend that you do not perform local data operations. However, the limits on the memory usage and CPU utilization do not apply to SQL or DataFrame nodes, except to `_pandas`, that are initiated by PyODPS.
- Functions may be limited in the following aspects due to the lack of packages such as `matplotlib`:
 - The use of the `plot` function of DataFrame is affected.
 - DataFrame user defined functions (UDFs) can be executed only after they are submitted to MaxCompute. As required by the Python sandbox, you can only use pure Python libraries and the NumPy library to execute UDFs. Other third-party libraries such as `pandas` cannot be used.
 - However, you can use the NumPy and `pandas` libraries that are pre-installed in DataWorks to execute non-UDFs. You are not allowed to use other third-party libraries that contain binary code.
- For compatibility reasons, `options.tunnel.use_instance_tunnel` is set to False in DataWorks by default. If you want to enable `InstanceTunnel` globally, you must set this parameter to True.
- For implementation reasons, the Python `atexit` package is not supported. You must use the `try-finally` structure to implement related features.

11.4. Basic operations

11.4.1. Overview

PyODPS supports basic operations on MaxCompute objects. You can use Python-compliant programming methods to perform operations on MaxCompute.

PyODPS allows you to perform basic operations on the following MaxCompute objects: projects, tables, SQL, task instances, resources, and functions.

11.4.2. Projects

This topic describes how to perform basic operations on projects by using PyODPS.

You can perform the following basic operations for a project:

- Obtain a project. You can call the `get_project()` method of a MaxCompute entry object to obtain a specific project.

```
project = o.get_project('my_project') # Obtain the specified project.
project = o.get_project()             # Obtain the current project.
```

The method requires a project name. If you do not specify a project name for the method, the method returns the current project.

- Check whether a project exists. You can call the `exist_project()` method to check whether a specific project exists.

11.4.3. Tables

This topic describes how to perform basic operations on tables by using PyODPS.

Basic operations

You can perform the following basic operations on tables:

- Obtain all tables in the current project by calling the `list_tables()` method of a MaxCompute entry object.

```
# Obtain all tables in the current project.
for table in o.list_tables():
```

- Check whether a specific table exists by calling the `exist_table()` method of a MaxCompute entry object.
- Obtain a specific table by calling the `get_table()` method of a MaxCompute entry object.

```
t = o.get_table('table_name')
t.schema
odps.Schema {
  c_int_a          bigint
  c_int_b          bigint
  c_double_a       double
  c_double_b       double
  c_string_a       string
  c_string_b       string
  c_bool_a         boolean
  c_bool_b         boolean
  c_datetime_a     datetime
  c_datetime_b     datetime
}
t.lifecycle
-1
print(t.creation_time)
2014-05-15 14:58:43
t.is_virtual_view
False
t.size
1408
t.comment
'table_name Table Comment'
t.schema.columns
[<column c_int_a, type bigint>,
 <column c_int_b, type bigint>,
 <column c_double_a, type double>,
 <column c_double_b, type double>,
 <column c_string_a, type string>,
 <column c_string_b, type string>,
 <column c_bool_a, type boolean>,
 <column c_bool_b, type boolean>,
 <column c_datetime_a, type datetime>,
 <column c_datetime_b, type datetime>]
t.schema['c_int_a']
<column c_int_a, type bigint>
t.schema['c_int_a'].comment
'Comment of column c_int_a'
```

You can obtain a table from another project by specifying the project parameter.

```
t = o.get_table('table_name', project='other_project')
```

Create a table schema

You can use one of the following methods to create a table schema:

- Create a schema based on table columns and optional partitions.

```

from odps.models import Schema, Column, Partition
columns = [Column(name='num', type='bigint', comment='the column'),
           Column(name='num2', type='double', comment='the column2')]
partitions = [Partition(name='pt', type='string', comment='the partition')]
schema = Schema(columns=columns, partitions=partitions)
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
schema.partitions
[<partition pt, type string>]
schema.names # Obtain the names of non-partition fields.
['num', 'num2']
schema.types # Obtain the data types of non-partition fields.
[bigint, double]

```

- Create a schema by calling the `Schema.from_lists()` method. This method is more convenient, but you cannot directly set comments for columns and partitions.

```

schema = Schema.from_lists(['num', 'num2'], ['bigint', 'double'], ['pt'], ['string'])
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]

```

Create a table

You can call the `create_table()` method to create a table in two ways:

- Use a table schema to create a table.

```

table = o.create_table('my_new_table', schema)
table = o.create_table('my_new_table', schema, if_not_exists=True) # Create the table only if no
table with the same name exists.
table = o.create_table('my_new_table', schema, lifecycle=7) # Set the lifecycle of the table.

```

- Create a table by specifying the names and data types of the fields to be contained in the table.

```

# Create a non-partitioned table.
table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
# Create a partitioned table with the specified table columns and partition fields.
table = o.create_table('my_new_table', ('num bigint, num2 double', 'pt string'), if_not_exists=True)

```

By default, when you create a table, you can use only the `BIGINT`, `DOUBLE`, `DECIMAL`, `STRING`, `DATETIME`, `BOOLEAN`, `MAP`, and `ARRAY` data types. If you need to use other data types such as `TINYINT` and `STRUCT`, you must set

`options.sql.use_odps2_extension` to `True`. Example:

```

from odps import options
options.sql.use_odps2_extension = True
table = o.create_table('my_new_table', 'cat smallint, content struct<title:varchar(100), body:string>')

```

Synchronize table updates

After another program updates a table, for example, the table schema, you can call the `reload()` method to synchronize the update.

```

table.reload()

```

Insert a record to a table

A record refers to a single row in a table. You can call the `new_record()` method to insert a record to a specific table.

```
t = o.get_table('mytable')
r = t.new_record(['val0', 'val1']) # The number of values must be equal to the number of fields in
the table schema.
r2 = t.new_record() # You can leave the value empty.
r2[0] = 'val0' # Set a value based on an offset.
r2['field1'] = 'val1' # Set a value based on the field name.
r2.field1 = 'val1' # Set a value based on an attribute.
print(record[0]) # Obtain the value at position 0.
print(record['c_double_a']) # Obtain a value based on a field.
print(record.c_double_a) # Obtain a value based on an attribute.
print(record[0: 3]) # Perform slicing operations.
print(record[0, 2, 3]) # Obtain values at multiple positions.
print(record['c_int_a', 'c_double_a']) # Obtain values based on multiple fields.
```

Obtain table data

You can use one of the following methods to obtain data from a table:

- Call the `read_table()` method of a MaxCompute entry object to read data from a table.

```
for record in o.read_table('test_table', partition='pt=test'):
    # Process a record.
```

- Call the `head()` method to obtain less than 10,000 data records from the beginning of a table.

```
t = o.get_table('table_name')
# Process each record.
for record in t.head(3):
```

- Call the `open_reader()` method.
 - Open the reader with a `WITH` clause.

```
with t.open_reader(partition='pt=test') as reader:
    count = reader.count
    for record in reader[5:10] # You can execute the statement multiple times until all records are
    read. The number of records is specified by count. You can change the code to parallel-operation
    code.
    # Process a record.
```

- Open the reader without a `WITH` clause.

```
reader = t.open_reader(partition='pt=test')
count = reader.count
for record in reader[5:10] # You can execute the statement multiple times until all records are
read. The number of records is specified by count. You can change the code to parallel-operation
code.
# Process a record.
```

Write data to a table

- Call the `write_table()` method of a MaxCompute entry object to write data to a table.

```
records = [[111, 'aaa', True],          # A list can be specified.
           [222, 'bbb', False],
           [333, 'ccc', True],
           [444, 'Chinese', False]]
o.write_table('test_table', records, partition='pt=test', create_partition=True)
```

Note

- Each time you call the `write_table()` method, MaxCompute generates a file on the server. This operation is time-consuming. In addition, if an excessive number of files are generated, the efficiency of subsequent queries is affected. We recommend that you write multiple records at a time or provide a generator object if you use the `write_table()` method.
- If you call the `write_table()` method to write data to a table, new data will be appended to existing data. PyODPS does not provide options to overwrite existing data. You must manually delete the data that you want to overwrite. For a non-partitioned table, you must call the `table.truncate()` method to delete data. For a partitioned table, you must delete partitions first and then create partitions again.

- Call the `open_writer()` method to write data to a table.

```
with t.open_writer(partition='pt=test') as writer:
records = [[111, 'aaa', True],          # A list can be specified.
           [222, 'bbb', False],
           [333, 'ccc', True],
           [444, 'Chinese', False]]
writer.write(records) # records can be iterable objects.
with t.open_writer(partition='pt1=test1,pt2=test2') as writer: # Write data in multi-level partitioning mode.
records = [t.new_record([111, 'aaa', True]), # Record objects can be used.
           t.new_record([222, 'bbb', False]),
           t.new_record([333, 'ccc', True]),
           t.new_record([444, 'Chinese', False])]
writer.write(records)
```

If the specified partition does not exist, set the `create_partition` parameter to `True` to create a partition.

Example:

```
with t.open_writer(partition='pt=test', create_partition=True) as writer:
records = [[111, 'aaa', True],          # A list can be specified.
           [222, 'bbb', False],
           [333, 'ccc', True],
           [444, 'Chinese', False]]
writer.write(records) # records can be iterable objects.
```

- Use multiple processes to concurrently write data to a table.

If multiple processes concurrently write data to a table, all processes use the same session ID but write data to different blocks. Each block corresponds to a file on the server. After all the processes finish writing data, the main process submits the data.

```

import random
from multiprocessing import Pool
from odps.tunnel import TableTunnel
def write_records(session_id, block_id):
    # Create a session with the specified session ID.
    local_session = tunnel.create_upload_session(table.name, upload_id=session_id)
    # Create a writer with the specified block ID.
    with local_session.open_record_writer(block_id) as writer:
        for i in range(5):
            # Generate data and write the data to the correct block.
            record = table.new_record([random.randint(1, 100), random.random()])
            writer.write(record)
if __name__ == '__main__':
    N_WORKERS = 3
    table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
    tunnel = TableTunnel(o)
    upload_session = tunnel.create_upload_session(table.name)
    # All processes use the same session ID.
    session_id = upload_session.id
    pool = Pool(processes=N_WORKERS)
    futures = []
    block_ids = []
    for i in range(N_WORKERS):
        futures.append(pool.apply_async(write_records, (session_id, i)))
        block_ids.append(i)
    [f.get() for f in futures]
    # Submit the data in all the blocks.
    upload_session.commit(block_ids)

```

Delete a table

You can call the `delete_table()` method to delete an existing table.

```

o.delete_table('my_table_name', if_exists=True) # Delete a table only if the table exists.
t.drop() # Call the drop() method to delete a table if the table exists.

```

Create a DataFrame

PyODPS provides a DataFrame framework, which allows you to conveniently query and manage MaxCompute data. You can call the `to_df()` method to convert a table to a DataFrame.

```

table = o.get_table('my_table_name')
df = table.to_df()

```

Manage partitions

- Check whether a table is partitioned.

```

if table.schema.partitions:
    print('Table %s is partitioned.' % table.name)

```

- Iterate over all the partitions in a table.

```

for partition in table.partitions:
    print(partition.name)
for partition in table.iterate_partitions(spec='pt=test'):
    # Iterate over level-2 partitions.

```

- Check whether a partition exists.

```
table.exist_partition('pt=test,sub=2015')
```

- Obtain a partition.

```
partition = table.get_partition('pt=test')
print(partition.creation_time)
2015-11-18 22:22:27
partition.size
0
```

- Create a partition.

```
t.create_partition('pt=test', if_not_exists=True) # Create a partition only if no partition with
the same name exists.
```

- Delete a partition.

```
t.delete_partition('pt=test', if_exists=True) # Delete a partition only if the partition exists.
partition.drop() # Call the drop() method to delete a partition if the partition exists.
```

Upload and download data by using MaxCompute Tunnel

MaxCompute Tunnel is the data tunnel of MaxCompute. You can use Tunnel to upload data to or download data from MaxCompute.

Note We recommend that you use the write and read interfaces of tables instead of Tunnel. In a Cython environment, PyODPS compiles C programs during installation to accelerate the Tunnel upload and download.

- Example of data upload

```
from odps.tunnel import TableTunnel
table = o.get_table('my_table')
tunnel = TableTunnel(odps)
upload_session = tunnel.create_upload_session(table.name, partition_spec='pt=test')
with upload_session.open_record_writer(0) as writer:
    record = table.new_record()
    record[0] = 'test1'
    record[1] = 'id1'
    writer.write(record)
    record = table.new_record(['test2', 'id2'])
    writer.write(record)
upload_session.commit([0])
```

- Example of data download

```
from odps.tunnel import TableTunnel
tunnel = TableTunnel(odps)
download_session = tunnel.create_download_session('my_table', partition_spec='pt=test')
with download_session.open_record_reader(0, download_session.count) as reader:
    for record in reader:
        # Process each record.
```

Note PyODPS does not support the upload of external tables.

11.4.4. SQL

This topic describes how to perform basic operations related to SQL statements by using PyODPS.

PyODPS supports MaxCompute SQL queries and provides methods to obtain execution results.

Execute SQL statements

You can call the `execute_sql()` and `run_sql()` methods of a MaxCompute entry object to execute SQL statements for creating task instances.

Parameter description:

- `statement`: the SQL statement to execute.
- `hints`: the runtime parameters. The `hints` parameter is of the DICT type.

Examples:

- Execute SQL statements.

```
o.execute_sql('select * from dual') # Execute the statement in synchronous mode. Other instances
are blocked until the SQL statement is executed.
instance = o.run_sql('select * from dual') # Execute the statement in asynchronous mode.
print(instance.get_logview_address()) # Obtain the Logview URL of an instance.
instance.wait_for_success() # Other instances are blocked until the SQL statement is executed.
```

- Set runtime parameters for SQL statements.

```
o.execute_sql('select * from pyodps_iris', hints={'odps.sql.mapper.split.size': 16})
```

If you set the `sql.settings` parameter globally, you need to configure runtime parameters each time you execute the statement.

```
from odps import options
options.sql.settings = {'odps.sql.mapper.split.size': 16}
o.execute_sql('select * from pyodps_iris') # The hints parameter is automatically set based on gl
obal settings.
```

Obtain the execution results of SQL statements

You can call the `open_reader` method to obtain execution results of SQL statements. When the query results are being read, the following situations may occur:

- The SQL statements return structured data.

```
with o.execute_sql('select * from dual').open_reader() as reader:
    for record in reader:
        # Process each record.
```

- If the `DESC` command is executed, you can use `reader.raw` to obtain the raw SQL query results.

```
with o.execute_sql('desc dual').open_reader() as reader:
    print(reader.raw)
```

If you specify `options.tunnel.use_instance_tunnel = True` when you call the `open_reader` method, PyODPS calls `InstanceTunnel` by default. However, if you are using an earlier version of MaxCompute or an error occurs when PyODPS calls `InstanceTunnel`, PyODPS generates an alert and automatically downgrades the call object to the old `Result` interface. You can identify the cause of the downgrade based on the alert information. If the result of `InstanceTunnel` does not meet your expectation, set this option to `False`. When you call the `open_reader` method, you can specify whether to call `InstanceTunnel` or the `Result` interface by setting the `tunnel` parameter.

```
# Call Instance Tunnel.
with o.execute_sql('select * from dual').open_reader(tunnel=True) as reader:
    for record in reader:
        # Process each record.
# Call the Result interface.
with o.execute_sql('select * from dual').open_reader(tunnel=False) as reader:
    for record in reader:
        # Process each record.
```

By default, PyODPS does not limit the amount of data that can be read from an instance. For protected projects, the amount of data that can be downloaded by using Tunnel is limited. If

`options.tunnel.limit_instance_tunnel` is not set, the limit is automatically enabled. The number of data entries that can be downloaded is limited based on project configurations. A maximum of 10,000 data entries can be downloaded. You can add a `limit` option to the `open_reader` method or set

```
options.tunnel.limit_instance_tunnel to True to manually limit the amount.
```

If your MaxCompute version only supports the old Result interface and you need to read all data, you can export the execution result of the SQL statement to another table and then use the table read interface to read data. This may be limited by project security settings.

In PyODPS V0.7.7.1 and later, you can use the `open_reader` method to call `InstanceTunnel` to obtain all data.

```
instance = o.execute_sql('select * from movielens_ratings limit 20000')
with instance.open_reader() as reader:
    print(reader.count)
    # for record in reader: Traverse the 20,000 data records. In this example, only 10 data records
    are obtained based on data slicing.
    for record in reader[:10]:
        print(record)
```

Set an alias

If the resource referenced by a UDF changes dynamically, you can set an alias for the old resource and use the alias as the name of the new resource. This way, you do not need to delete the UDF or create another UDF.

```

from odps.models import Schema
myfunc = '''\
from odps.udf import annotate
from odps.distcache import get_cache_file
@annotate('bigint->bigint')
class Example(object):
    def __init__(self):
        self.n = int(get_cache_file('test_alias_res1').read())
    def evaluate(self, arg):
        return arg + self.n
'''
res1 = o.create_resource('test_alias_res1', 'file', file_obj='1')
o.create_resource('test_alias.py', 'py', file_obj=myfunc)
o.create_function('test_alias_func',
                 class_type='test_alias.Example',
                 resources=['test_alias.py', 'test_alias_res1'])
table = o.create_table(
    'test_table',
    schema=Schema.from_lists(['size'], ['bigint']),
    if_not_exists=True
)
data = [[1, ], ]
# Write a row of data that contains only one value: 1.
o.write_table(table, 0, [table.new_record(it) for it in data])
with o.execute_sql(
    'select test_alias_func(size) from test_table').open_reader() as reader:
    print(reader[0][0])
res2 = o.create_resource('test_alias_res2', 'file', file_obj='2')
# Set the alias of resource res1 as the name of resource res2 without the need to modify the UDF or
resource.
with o.execute_sql(
    'select test_alias_func(size) from test_table',
    aliases={'test_alias_res1': 'test_alias_res2'}).open_reader() as reader:
    print(reader[0][0])

```

Execute SQL statements in an interactive environment

In IPython and Jupyter, SQL plug-ins can be used to execute SQL statements and parameterized queries are supported.

Set biz_id

Occasionally, you must specify `biz_id` for the SQL statement to execute. Otherwise, an error occurs. In this case, you can set `biz_id` in the global options to troubleshoot the error.

```

from odps import options
options.biz_id = 'my_biz_id'
o.execute_sql('select * from pyodps_iris')

```

11.4.5. Task instances

This topic describes how to perform basic operations on task instances by using PyODPS.

Basic operations

Tasks are implemented as MaxCompute instances. You can call `list_instances` to retrieve all the instances in the project. You can use `exist_instance` to determine if an instance exists, and use `get_instance` to retrieve instances.

```
for instance in o.list_instances():
    print(instance.id)
    o.exist_instance('my_instance_id')
```

You can call the stop method for an instance or call `stop_instance` at the MaxCompute entry to stop an instance.

Obtain the Logview URL of an instance

For an SQL task instance, you can call the `get_logview_address` method to obtain its Logview URL.

```
# Obtain the Logview URL based on an existing instance object.
instance = o.run_sql('desc pyodps_iris')
print(instance.get_logview_address())
# Obtain the Logview URL based on an instance ID.
instance = o.get_instance('2016042605520945g9k5pvyi2')
print(instance.get_logview_address())
```

For an XFlow task instance, you must enumerate its sub-instances and obtain the Logview URL of each sub-instance.

```
instance = o.run_xflow('AppendID', 'algo_public', {'inputTableName': 'input_table', 'outputTableName': 'output_table'})
for sub_inst_name, sub_inst in o.get_xflow_sub_instances(instance).items():
    print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address()))
```

Obtain the status of an instance

An instance can be in the Running, Suspended, or Terminated state. You can obtain the status of an instance from the status property. You can call the `is_terminated` method to check whether the execution of the current instance is complete and call the `is_successful` method to check whether the execution is successful. If the task is running or fails, False is returned for both methods.

```
instance = o.get_instance('2016042605520945g9k5pvyi2')
instance.status
<Status.TERMINATED: 'Terminated'>
from odps.models import Instance
instance.status == Instance.Status.TERMINATED
True
instance.status.value
'Terminated'
```

You can call the `wait_for_completion` method to ask the system to return the status until the execution of the instance is completed. The `wait_for_success` method functions similarly. The difference is that if the execution fails, an exception is reported.

Perform operations on sub-instances

A running instance may contain one or more sub-instances.

You can call the `get_task_names` method to obtain the names of all sub-instances.

```
instance.get_task_names()
['SQLDropTableTask']
```

After you obtain a sub-instance name, you can call `get_task_result` to obtain the execution result of the sub-instance. This method returns the execution result of each sub-instance in the form of a dictionary.

```
instance = o.execute_sql('select * from pyodps_iris limit 1')
instance.get_task_names()
['AnonymousSQLTask']
instance.get_task_result('AnonymousSQLTask')
'"sepallength","sepalwidth","petallength","petalwidth","name"\n5.1,3.5,1.4,0.2,"Iris-setosa"\n'
instance.get_task_results()
OrderedDict([('AnonymousSQLTask', "sepallength","sepalwidth","petallength","petalwidth","name"\n5.1,
3.5,1.4,0.2,"Iris-setosa"\n'))
```

You can call `get_task_progress` to obtain the current running progress of a specific sub-instance when the instance is running.

```
while not instance.is_terminated():
    for task_name in instance.get_task_names():
        print(instance.id, instance.get_task_progress(task_name).get_stage_progress_formatted_string
())
        time.sleep(10)
20190519101349613gzbzufck2 2019-05-19 18:14:03 M1_Stg1_job0:0/1/1[100%]
```

11.4.6. Resources

This topic describes how to perform basic operations on resources by using PyODPS.

PyODPS supports file and table resources.

Resources commonly apply to UDFs and MapReduce on MaxCompute.

You can call the `list_resources` method to query all resources, the `exist_resource` method to check whether a resource exists, and the `delete_resource` method to delete a resource. You can also call the `drop` method to delete a resource.

Manage file resources

PyODPS supports basic file types, and the .py, .jar, and .archive types.

- Create a file resource

You can specify a resource name, a file type, a file-like object or a string object to create a file resource.

```
resource = o.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file')) # Use
a file-like object to create a file resource.
resource = o.create_resource('test_py_resource', 'py', file_obj='import this') # Use a string to
create a file resource.
```

- Read and modify a file resource

You can call the `open` method for a file resource or call the `open_resource` method at the MaxCompute entry to open a file resource. The opened object is a file-like object. Similar to the `open` method built in Python, file resources also support various opening modes.

```

with resource.open('r') as fp: # Open the specified file in read mode.
    content = fp.read() # Read all content.
    fp.seek(0) # Return to the beginning of the file.
    lines = fp.readlines() # Read multiple lines.
    fp.write('Hello World') # Error. Data cannot be written to a file in read mode.
with o.open_resource('test_file_resource', mode='r+') as fp: # Open the file in read/write mode.
    fp.read()
    fp.tell() # Locate the current position.
    fp.seek(10)
    fp.truncate() # Truncate the file to the specified length.
    fp.writelines(['Hello\n', 'World\n']) # Write multiple lines to the file.
    fp.write('Hello World')
    fp.flush() # Manually call the method to submit the update to MaxCompute.

```

PyODPS supports the following opening modes:

- **r**: read mode. The file can be opened, but data cannot be written to it.
- **w**: write mode. Data can be written to the file, but data in the file cannot be read. If a file is opened in write mode, the file content is cleared first.
- **a**: append mode. Data can be added to the end of the file.
- **r+**: read/write mode. You can read data from and write data to the file.
- **w+**: This mode is similar to the **r+** mode. The only difference is that the file content is cleared first.
- **a+**: This mode is similar to the **r+** mode. The only difference is that data can be written only to the end of the file.

In PyODPS, file resources can be opened in binary mode. For example, some compressed files must be opened in binary mode. **rb** indicates that a file is opened in binary read mode, and **r+b** indicates that a file is opened in binary read/write mode.

Manage table resources

- Create a table resource

```
o.create_resource('test_table_resource', 'table', table_name='my_table', partition='pt=test')
```

- Update a table resource

```
table_resource = o.get_resource('test_table_resource')
table_resource.update(partition='pt=test2', project_name='my_project2')
```

- Obtain a table and a partition

```
table_resource = o.get_resource('test_table_resource')
table = table_resource.table
print(table.name)
partition = table_resource.partition
print(partition.spec)
```

- Read data from and write data to a table

```
table_resource = o.get_resource('test_table_resource')
with table_resource.open_writer() as writer:
    writer.write([0, 'aaaa'])
    writer.write([1, 'bbbb'])
with table_resource.open_reader() as reader:
    for rec in reader:
        print(rec)
```

11.4.7. Functions

This topic describes how to perform basic operations on functions by using PyODPS.

You can create user defined functions (UDFs) and use them in MaxCompute SQL.

Create a function

You can call the `create_function()` method of a MaxCompute entry object to create a function. Example:

```
# Reference a resource in the current project.
resource = o.get_resource('my_udf.py')
function = o.create_function('test_function', class_type='my_udf.Test', resources=[resource])
# Reference a resource in another project.
resource2 = o.get_resource('my_udf.py', project='another_project')
function2 = o.create_function('test_function2', class_type='my_udf.Test', resources=[resource2])
```

Delete a function

You can call the `delete_function()` method of a MaxCompute entry object to delete a function. You can also call the `drop()` method to delete a function. Example:

```
o.delete_function('test_function')
function.drop() # Call the drop() method if the function exists.
```

Update a function

You can call the `update()` method to update a function. Example:

```
function = o.get_function('test_function')
new_resource = o.get_resource('my_udf2.py')
function.class_type = 'my_udf2.Test'
function.resources = [new_resource, ]
function.update() # Update the function.
```

11.5. DataFrame

This topic describes how to create and manage a DataFrame object. This topic also provides information about how to use DataFrame to process data.

PyODPS provides an interface similar to pandas called PyODPS DataFrame. This interface operates on MaxCompute tables and makes full use of the capabilities of MaxCompute. You can also change the data source from MaxCompute tables to pandas DataFrame. This way, the same code can be executed on pandas.

For the complete DataFrame document, see [DataFrame](#).

DataFrame object operations

The following example demonstrates how to create a DataFrame object.

 **Note** The data used in the example is downloaded from [movielens 100K](#). You can download the data as needed.

Assume that the following tables exist: `pyodps_ml_100k_movies` that contains movie-related data, `pyodps_ml_100k_users` that contains user-related data, and `pyodps_ml_100k_ratings` that contains rating-related data.

1. If no MaxCompute object is provided in the runtime environment, you must create an object.

```
from odps import ODPS
o = ODPS('**your-access-id**', '**your-secret-access-key**',
        project='**your-project**', endpoint='**your-end-point**')
```

2. You need only to specify a table object to create a DataFrame object.

```
from odps.df import DataFrame
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

3. You can use the dtypes attribute to view the fields of the DataFrame object and the data types of the fields.

```
users.dtypes
odps.Schema {
  user_id      int64
  age          int64
  sex          string
  occupation   string
  zip_code     string
}
```

4. You can use the `head` method to have a quick preview on the first N data records.

```
users.head(10)
  user_id  age  sex  occupation  zip_code
0         1   24   M   technician  85711
1         2   53   F         other  94043
2         3   23   M         writer  32067
3         4   24   M   technician  43537
4         5   33   F         other  15213
5         6   42   M   executive  98101
6         7   57   M administrator  91344
7         8   36   M administrator  05201
8         9   29   M         student  01002
9        10   53   M         lawyer  90703
```

5. If you do not want to view all the fields, you can perform the following operations as needed:

- Specify the fields that you want to query.

```
users[['user_id', 'age']].head(5)
  user_id  age
0         1   24
1         2   53
2         3   23
3         4   24
4         5   33
```

- Exclude several fields. Example:

```
users.exclude('zip_code', 'age').head(5)
  user_id  sex  occupation
0         1   M   technician
1         2   F         other
2         3   M         writer
3         4   M   technician
4         5   F         other
```

- Exclude some fields and add new fields based on computation. For example, add the `sex_bool` attribute

and set it to True if sex is M. Otherwise, set it to False.

```
users.select(users.exclude('zip_code', 'sex'), sex_bool=users.sex == 'M').head(5)
  user_id  age  occupation  sex_bool
0       1   24  technician    True
1       2   53     other    False
2       3   23     writer    True
3       4   24  technician    True
4       5   33     other    False
```

6. You can obtain the numbers of male and female users.

```
users.groupby(users.sex).agg(count=users.count())
  sex  count
0    F   273
1    M   670
```

7. To divide users by occupation, you can obtain the first 10 occupations that have the largest population, and sort the occupations in descending order of population.

```
df = users.groupby('occupation').agg(count=users['occupation'].count())
df.sort(df['count'], ascending=False)[:10]
  occupation  count
0     student   196
1      other   105
2   educator    95
3 administrator    79
4    engineer    67
5  programmer    66
6  librarian    51
7     writer    45
8  executive    32
9  scientist    31
```

Alternatively, you can use the `value_counts` method. The number of records returned by this method is limited by `options.df.odps.sort.limit`.

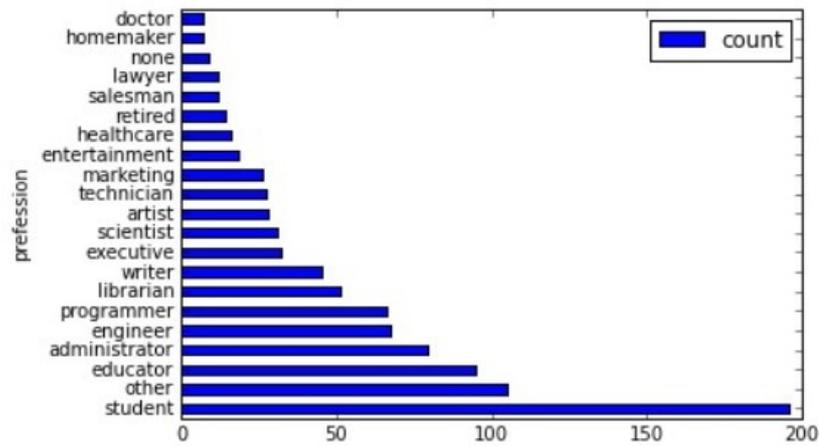
```
users.occupation.value_counts()[:10]
  occupation  count
0     student   196
1      other   105
2   educator    95
3 administrator    79
4    engineer    67
5  programmer    66
6  librarian    51
7     writer    45
8  executive    32
9  scientist    31
```

8. Show data in a more intuitive graph.

```
%matplotlib inline
```

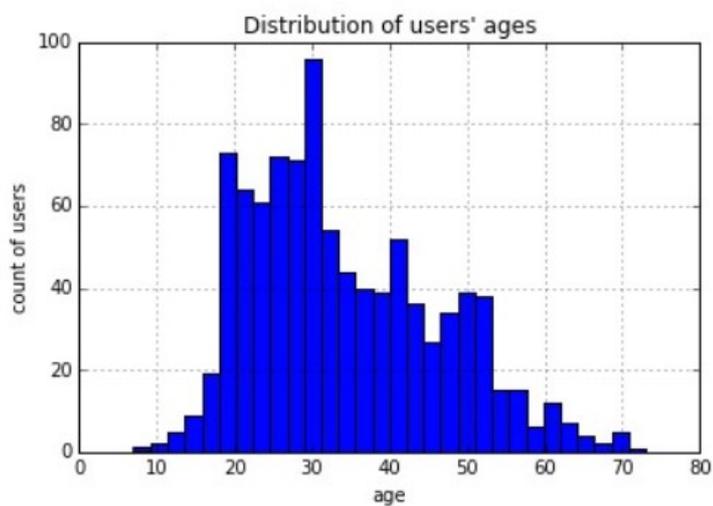
9. Use a horizontal column chart to visualize data.

```
users['occupation'].value_counts().plot(kind='barh', x='occupation', ylabel='profession')
<matplotlib.axes._subplots.AxesSubplot at 0x10653cfd0>
```



10. Divide users into 30 groups by age and view the histogram of age distribution.

```
users.age.hist(bins=30, title="Distribution of users' ages", xlabel='age', ylabel='count of users')
<matplotlib.axes._subplots.AxesSubplot at 0x10667a510>
```



11. Use `join` to join the three tables and save them as a new table.

```

movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
o.delete_table('pyodps_ml_100k_lens', if_exists=True)
lens = movies.join(ratings).join(users).persist('pyodps_ml_100k_lens')
lens.dtypes
odps.Schema {
  movie_id          int64
  title             string
  release_date      string
  video_release_date string
  imdb_url          string
  user_id           int64
  rating            int64
  unix_timestamp    int64
  age               int64
  sex               string
  occupation        string
  zip_code          string
}

```

12. Divide users aged 0 to 80 into eight age groups.

```

labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
cut_lens = lens[lens, lens.age.cut(range(0, 81, 10), right=False, labels=labels).rename('age_group')]

```

13. View the first 10 data records of a single age in a group.

```

cut_lens['age_group', 'age'].distinct()[:10]

```

	age_group	age
0	0-9	7
1	10-19	10
2	10-19	11
3	10-19	13
4	10-19	14
5	10-19	15
6	10-19	16
7	10-19	17
8	10-19	18
9	10-19	19

14. View the total rating and average rating of users in each age group.

```

cut_lens.groupby('age_group').agg(cut_lens.rating.count().rename('total_rating'), cut_lens.rating.mean().rename('avg_rating'))

```

	age_group	avg_rating	total_rating
0	0-9	3.767442	43
1	10-19	3.486126	8181
2	20-29	3.467333	39535
3	30-39	3.554444	25696
4	40-49	3.591772	15021
5	50-59	3.635800	8704
6	60-69	3.648875	2623
7	70-79	3.649746	197

DataFrame data processing

The following example demonstrates how to process DataFrame data.

 **Note** The data used in the example is downloaded from [Iris dataset](#). You can download the data as needed.

1. Create a test data table.

Use the table management feature of DataWorks to create a table, and then click **DDL mode**.

Enter the CREATE TABLE statement and submit the table. Example:

```
CREATE TABLE `pyodps_iris` (
  `sepallength` double COMMENT 'sepallength(cm)',
  `sepalwidth` double COMMENT 'sepalwidth(cm)',
  `petallength` double COMMENT 'petallength(cm)',
  `petalwidth` double COMMENT 'petalwidth(cm)',
  `name` string COMMENT 'name'
) ;
```

2. Upload test data.

Click the **Import** icon.

Enter the table name and upload the downloaded dataset.

Select **Match by location** and click **Import Data**.

3. Create a PyODPS node to store and run code.

4. Enter the code and click the Run icon. You can view the result in the **Runtime Log** section in the lower pane.

Code details:

```
from odps.df import DataFrame
from odps.df import output
iris = DataFrame(o.get_table('pyodps_iris')) # Create the DataFrame object iris from the MaxCompute table.
print iris.head(10)
print iris.sepallength.head(5) # Display part of the iris content.
# Use a user defined function to calculate the sum of two columns of iris.
print iris.apply(lambda row: row.sepallength + row.sepalwidth, axis=1, reduce=True, types='float').rename('sepaladd').head(3)
# Specify the output name and type of the function.
@output(['iris_add', 'iris_sub'], ['float', 'float'])
def handle(row):
    # Use the yield keyword to return multiple rows of results.
    yield row.sepallength - row.sepalwidth, row.sepallength + row.sepalwidth
    yield row.petallength - row.petalwidth, row.petallength + row.petalwidth
# Display the results of the first five rows. axis=1 indicates that the axis of the column extends horizontally.
print iris.apply(handle, axis=1).head(5)
```

11.6. User experience enhancement

11.6.1. Command line

This topic describes command line enhancement.

PyODPS provides an enhanced command line tool. You can perform the following steps to configure and call the tool:

1. Import the PyODPS enhancement tool.

```
from odps.inter import setup, enter, teardown
```

2. Configure your account.

```
setup('**your-access-id**', '**your-access-key**', '**your-project**', endpoint='**your-endpoint**')
```

Note

- If you do not specify the room parameter, the default room is used.
- After you configure an account, you do not need to enter the account information again.

3. Call the enter method to create a room object on a Python interactive interface.

```
room = enter()
o = room.odps
o.get_table('dual')
odps.Table
name: odps_test_sqltask_finance.`dual`
schema:
  c_int_a          : bigint
  c_int_b          : bigint
  c_double_a       : double
  c_double_b       : double
  c_string_a       : string
  c_string_b       : string
  c_bool_a         : boolean
  c_bool_b         : boolean
  c_datetime_a     : datetime
  c_datetime_b     : datetime
```

 **Note** The MaxCompute entry object is not automatically updated when you change the setup of the room. You must call `enter()` again to retrieve the new room object.

After you configure an account and call objects, you can store, retrieve, or delete objects in the room or delete the entire room.

- You can store commonly used MaxCompute tables or resources in the room. Example:

```
room.store('stored-table', o.get_table('dual'), desc='Simple stored table example')
```

- You can call the `display` method to display the stored objects as a table. Example:

```
room.display()
```

Result:

```
default name      desc
stored-table      Simple stored table example
iris              Iris dataset
```

- You can use `room['stored-table']` or `room.iris` to retrieve the stored objects. Example:

```
room['stored-table']
odps.Table
  name: odps_test_sqltask_finance.`dual`
  schema:
    c_int_a      : bigint
    c_int_b      : bigint
    c_double_a   : double
    c_double_b   : double
    c_string_a   : string
    c_string_b   : string
    c_bool_a     : boolean
    c_bool_b     : boolean
    c_datetime_a : datetime
    c_datetime_b : datetime
```

- You can call the `drop` method to delete objects in the room. Example:

```
room.drop('stored-table')
room.display()
default name      desc
iris              Iris dataset
```

- You can call the `teardown` method to delete a room. If no parameters are specified, the default room is deleted.

```
teardown()
```

11.6.2. IPython

This topic describes IPython enhancement.

PyODPS provides the IPython plug-in to facilitate MaxCompute operations.

IPython enhancement

Some commands are provided for command line enhancement.

- You can run the following code to load the plug-in:

```
%load_ext odps
%enter
```

The following result is returned:

```
<odps.inter.Room at 0x11341df10>
```

- In this case, the global `o` and `odps` variables can be retrieved. You can run the `o.get_table` or `odps.get_table` command to call a table.

```
o.get_table('dual')
odps.get_table('dual')
```

The following result is returned:

```
odps.Table
  name: odps_test_sqltask_finance.`dual`
  schema:
    c_int_a      : bigint
    c_int_b      : bigint
    c_double_a   : double
    c_double_b   : double
    c_string_a   : string
    c_string_b   : string
    c_bool_a     : boolean
    c_bool_b     : boolean
    c_datetime_a : datetime
    c_datetime_b : datetime
```

- You can run the following command to display the stored objects as a table:

```
%stores
```

The following result is returned:

```
default name      desc
iris              Iris dataset
```

Object name completion

PyODPS enhances the code completion feature that is provided by IPython. When you write a statement such as `o.get_XXX`, the object name is automatically completed. In the following examples, `<tab>` is used to denote pressing the Tab key. When you enter the statement and encounter `<tab>`, press the Tab key.

- You can use the Tab key to complete the object name.

```
o.get_table(<tab>
```

- You can enter the first few characters of an object name and press the Tab key to complete it:

```
o.get_table('tabl<tab>
```

IPython auto-completes the table name that starts with tabl.

- This feature also completes the names of objects in different projects. Syntax:

```
o.get_table(project='project_name', name='tabl<tab>')
o.get_table('tabl<tab>', project='project_name')
```

- If multiple matching objects exist, IPython provides a list. `options.completion_size` specifies the maximum number of objects in the list. The default value is 10.

SQL statements

PyODPS provides an SQL plug-in to execute MaxCompute SQL statements.

- You can use `%sql` to execute a single-line SQL statement. Example:

```
In [*]: %sql select * from pyodps_iris limit 5
|=====| 1 / 1 (100.00%) 3s
Out[*]:
   sepallength  sepalwidth  petallength  petalwidth  name
0           5.1           3.5           1.4           0.2  Iris-setosa
1           4.9           3.0           1.4           0.2  Iris-setosa
2           4.7           3.2           1.3           0.2  Iris-setosa
3           4.6           3.1           1.5           0.2  Iris-setosa
4           5.0           3.6           1.4           0.2  Iris-setosa
```

- You can use `%sql` to execute a multiple-line SQL statement. Example:

```
In [*]: %%sql
....: select * from pyodps_iris
....: where sepallength < 5
....: limit 5
....:
|=====| 1 / 1 (100.00%) 15s
Out[*]:
   sepallength  sepalwidth  petallength  petalwidth  name
0           4.9           3.0           1.4           0.2  Iris-setosa
1           4.7           3.2           1.3           0.2  Iris-setosa
2           4.6           3.1           1.5           0.2  Iris-setosa
3           4.6           3.4           1.4           0.3  Iris-setosa
4           4.4           2.9           1.4           0.2  Iris-setosa
```

- To execute parameterized SQL statements, you can use `:parameter` to specify the parameter. Example:

```
In [1]: %load_ext odps
In [2]: mytable = 'dual'
In [3]: %sql select * from :mytable
|=====| 1 / 1 (100.00%) 2s
Out[3]:
   c_int_a  c_int_b  c_double_a  c_double_b  c_string_a  c_string_b  c_bool_a  \
0         0         0       -1203           0           0       -1203   True
   c_bool_b  c_datetime_a  c_datetime_b
0  False  2012-03-30 23:59:58  2012-03-30 23:59:59
```

- For SQL runtime parameters, you can use `%set` to set a global parameter or use `SET` within an SQLCell to set a local parameter. The following example sets a local parameter, which does not affect the global setting.

```
In [*]: %%sql
       set odps.sql.mapper.split.size = 16;
       select * from pyodps_iris;
```

- The following example sets a global parameter. Global settings apply to all subsequent SQL statements.

```
In [*]: %set odps.sql.mapper.split.size = 16
```

Upload pandas DataFrame to MaxCompute tables

PyODPS provides commands to upload pandas DataFrame objects to MaxCompute tables.

You need only to use the `%persist` command. The first parameter `df` is the variable name. The second parameter `pyodps_pandas_df` is the MaxCompute table name.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.arange(9).reshape(3, 3), columns=list('abc'))
%persist df pyodps_pandas_df
```

11.6.3. Jupyter Notebook

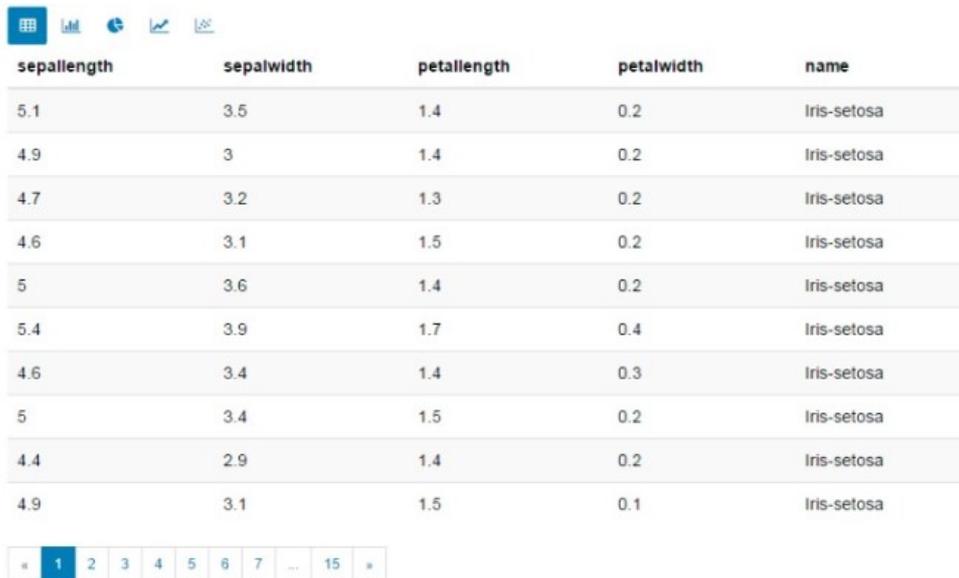
This topic describes Jupyter Notebook enhancement.

PyODPS enhances the result exploration and progress display features of Jupyter Notebook.

Result exploration

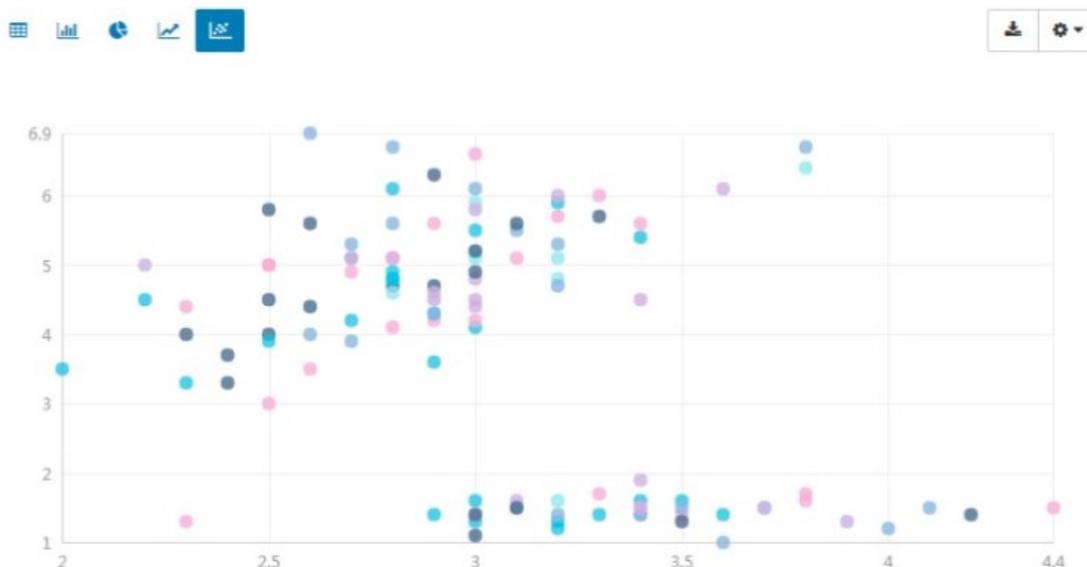
PyODPS provides a data exploration feature in Jupyter Notebook for SQLCell and DataFrame. You can use interactive data exploration tools to browse local data and create graphs.

1. If the execution result is a DataFrame object, PyODPS reads the result and displays it in a paged table. You can click a page number, the Previous button, or the Next button to browse the data.

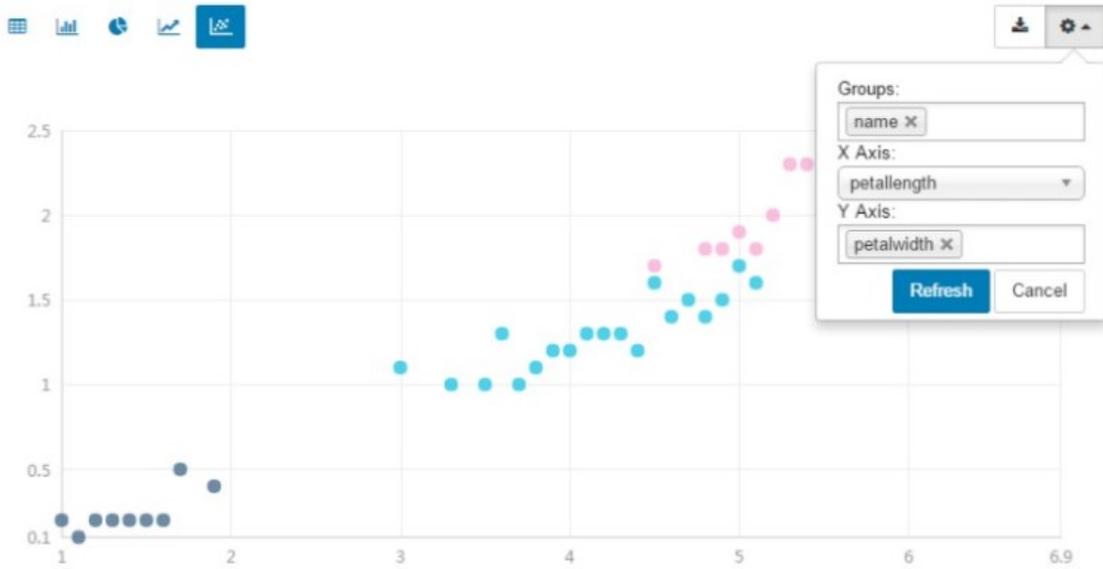


sepalength	sepalwidth	petallength	petalwidth	name
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa

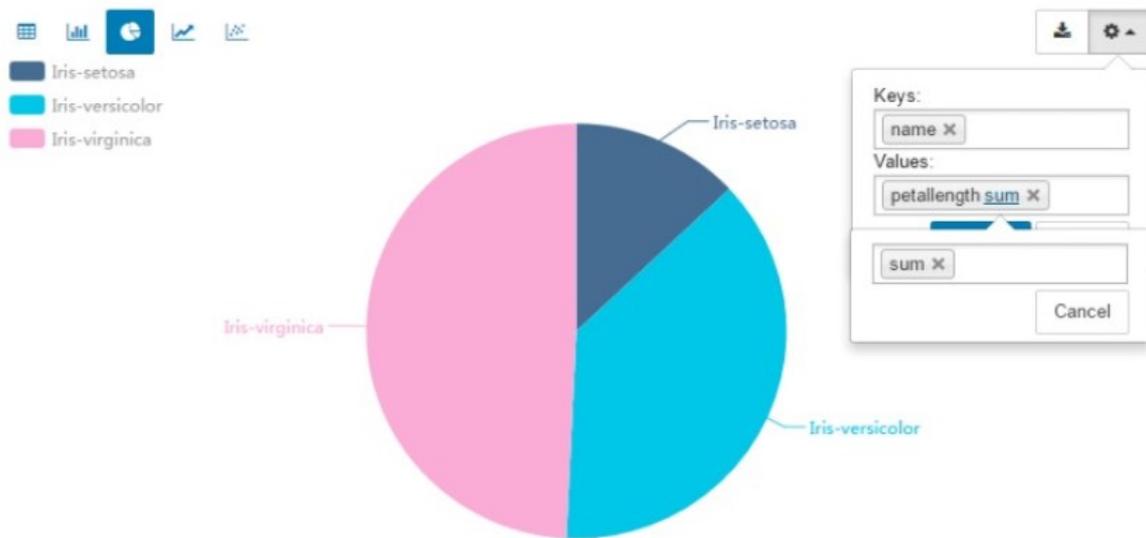
2. You can select other display modes on the top of the table to display the result in a column chart, pie chart, line chart, or scatter chart. The following figure shows a scatter chart created based on the default fields, which are the first three fields.



3. You can click the Settings icon in the upper-right corner of a graph to modify the settings. For example, set Groups to name, X Axis to petallength, and Y Axis to petalwidth. The result is shown in the following graph. The petallength-petalwidth settings display the data in a manner that is easy to understand.



For column charts and pie charts, you can select an aggregate function for the value fields. The default aggregate function for column charts is sum, and that for pie charts is count. You can click the function name next to the name of the value field to select another function. For line charts, the values on the x-axis cannot be null. If a value is null, the graph may not be correctly displayed.



4. After you make the graph, click the **Download** icon to save it.

Note To use this feature, you must install pandas and ipywidgets.

Progress display

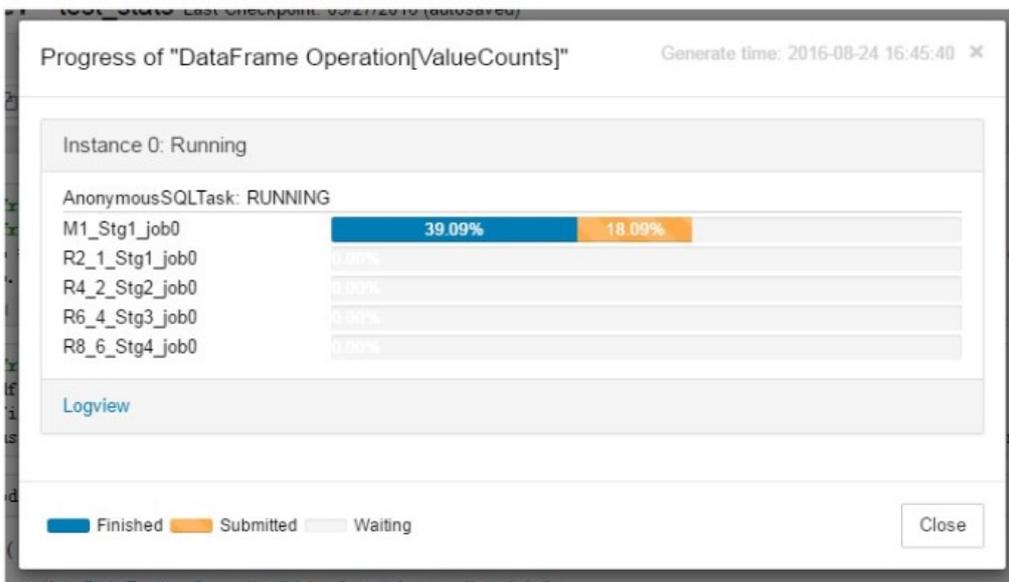
The execution of large jobs takes extended periods of time. PyODPS provides progress bars to show the execution progress. When DataFrame jobs, machine learning jobs, or SQL statements that start with `%sql` are executed in Jupyter Notebook, a list of these jobs and their overall progress are displayed.

```
In [*]: from odps import ODPS
        from odps.df.examples import create_ionosphere
        o = ODPS(access_id, secret_access_key, project=project, endpoint=endpoint)
        df = create_ionosphere(o)['a01', 'a02', 'a03', 'a04', 'class']
        df.calc_summary()
```

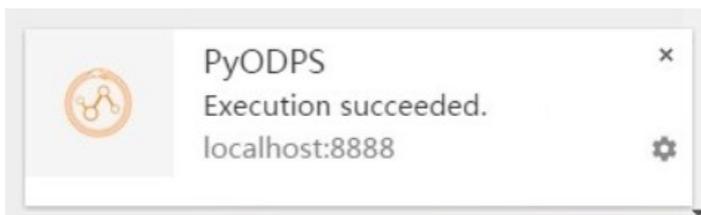
✕ (50.00%) 

Executing: [summary](#)

When you click a job name, a dialog box that displays the progress of each task in the job appears.



After the execution has been completed, a message that indicates whether the job is successful appears.



11.7. Configurations

This topic describes the configuration options provided by PyODPS.

PyODPS provides a series of configuration options. You can obtain them by using `odps.options`. Example:

```

from odps import options
# Set the lifecycle option to specify the lifecycle of all output tables.
options.lifecycle = 30
# Set the tunnel.string_as_binary option to True to use bytes instead of Unicode to download data of
the STRING type.
options.tunnel.string_as_binary = True
# When you execute PyODPS DataFrames in MaxCompute, you can refer to the following configuration to
set the limit to a relatively large value during a sort operation.
options.df.odps.sort.limit = 100000000

```

General configurations

Option	Description	Default value
end_point	The endpoint of MaxCompute.	None
default_project	The default project.	None
log_view_host	The hostname of Logview.	None
log_view_hours	The retention time of Logview. Unit: hours.	24
local_timezone	The time zone that is used. True indicates local time, and False indicates UTC. The time zone of pytz can also be used.	None
lifecycle	The lifecycle of all tables.	None
temp_lifecycle	The lifecycle of temporary tables.	1
biz_id	The user ID.	None
verbose	Specifies whether to display logs.	False
verbose_log	The log receiver.	None
chunk_size	The size of the write buffer.	1496
retry_times	The number of request retries.	4
pool_connections	The number of cached connections in the connection pool.	10
pool_maxsize	The maximum capacity of the connection pool.	10
connect_timeout	The connection timeout period.	5
read_timeout	The read timeout period.	120
api_proxy	The API proxy server.	None
data_proxy	The data proxy server.	None
completion_size	The limit on the number of object completion listing items.	10

Option	Description	Default value
notebook_repr_widget	Specifies whether to use interactive graphs.	True
sql.settings	Global hints for MaxCompute SQL.	None
sql.use_odps2_extension	Specifies whether to enable MaxCompute 2.0 language extension.	False

Data upload and download configurations

Option	Description	Default value
tunnel.endpoint	The endpoint of MaxCompute Tunnel.	None
tunnel.use_instance_tunnel	Specifies whether to use InstanceTunnel to obtain execution results.	True
tunnel.limit_instance_tunnel	Specifies whether to limit the number of data records obtained by using InstanceTunnel.	None
tunnel.string_as_binary	Specifies whether to use bytes instead of Unicode for data of the STRING type.	False

DataFrame configurations

Option	Description	Default value
interactive	Specifies whether DataFrames are used in an interactive environment.	Depending on the detection value
df.analyze	Specifies whether to enable functions that are not built in MaxCompute.	True
df.optimize	Specifies whether to enable full DataFrame optimization.	True
df.optimizes.pp	Specifies whether to enable DataFrame predicate pushdown optimization.	True
df.optimizes.cp	Specifies whether to enable DataFrame column pruning optimization.	True
df.optimizes.tunnel	Specifies whether to enable DataFrame tunnel optimization.	True

Option	Description	Default value
df.quote	Specifies whether to use a pair of grave accents (` `) to mark field and table names in the backend of MaxCompute SQL.	True
df.libraries	The resource name of the third-party library that is used for DataFrame operations.	None
df.supersede_libraries	Specifies whether to use the self-uploaded NumPy to replace the version in the service.	False
df.odps.sort.limit	The default limit on the number of items that are added during a sort operation of DataFrames.	10000

Machine learning configurations

Option	Description	Default value
ml.xflow_settings	The XFlow execution configuration.	None
ml.xflow_project	The default XFlow project name.	algo_public
ml.use_model_transfer	Specifies whether to use ModelTransfer to obtain the Predictive Model Markup Language (PMML) files of models.	False
ml.model_volume	The name of the volume used by ModelTransfer.	pyodps_volume

11.8. API overview

This topic provides the links to PyODPS API documentation, which provides parameter descriptions and examples of each function:

- [Definitions](#)
- [DataFrame Reference](#)

11.9. FAQ

This topic provides the best practices and some frequently asked questions about PyODPS, which help you efficiently develop PyODPS programs.

How do I view the current PyODPS version?

Run the following commands:

```
import odps
print(odps.__version__)
```

How do I troubleshoot the "Project not found" error?

This problem is generally caused by invalid endpoint configurations. You must check the configurations and rectify the problem. You also need to check whether the position of the MaxCompute object parameter is correct.

How do I manually specify a Tunnel endpoint?

You can create your MaxCompute object with the `tunnel_endpoint` parameter specified, as shown in the following code. Replace the content enclosed with asterisks (*) with actual parameter values and remove the asterisks (*).

```
from odps import ODPS
o = ODPS('**your-access-id**', '**your-secret-access-key**', '**your-default-project**',
        endpoint='**your-end-point**', tunnel_endpoint='**your-tunnel-endpoint**')
```

How do I troubleshoot the "project is protected" error reported while data is being read?

The project security policy does not allow you to read data from tables. To retrieve all the data, you can use the following solutions:

- Contact the project owner to add exception rules.
- Use DataWorks or other masking tools to mask the data and then export the data to an unprotected project before reading it.

To retrieve part of the data, you can use the following solutions:

- Use `o.execute_sql('select * from <table_name>').open_reader()` .
- Use `o.get_table('<table_name>').to_df()` in DataFrame.

I can only retrieve a maximum of 10,000 data records by executing SQL command `open_reader`. How do I retrieve more than 10,000 data records?

Use `create table as select ...` to save the SQL result to a table, and then use `table.open_reader` to read data.

How do I troubleshoot the "ODPSError: ODPS entrance should be provided" error reported when I upload pandas DataFrame to MaxCompute?

This error is reported because the global MaxCompute object cannot be found. Use one of the following methods to resolve this problem:

- If you use the room mechanism `%enter` , configure the global MaxCompute object.
- Call the `to_global` method for the MaxCompute object.
- Use the following MaxCompute parameter:

```
DataFrame(pd_df).persist('your_table', odps=odps)
```

How do I use `max_pt` in DataFrame?

Use the `odps.df.func` module to call built-in functions of MaxCompute. Example:

```
from odps.df import func
df = o.get_table('your_table').to_df()
df[df.ds == func.max_pt('your_project.your_table')] # ds is a partition column.
```

How do I troubleshoot the "table lifecycle is not specified in mandatory mode" error reported when I use DataFrame to write data to a table?

Your project requires that every table be created with a lifecycle. Therefore, you must make the following configuration every time you run your own code:

```
from odps import options
options.lifecycle = 7 # You can also set this parameter to your expected lifecycle in days.
```

How do I traverse each row of data in PyODPS DataFrame?

PyODPS DataFrame does not support this feature. PyODPS DataFrame focuses on handling large volumes of data. Traversing data is inefficient.

We recommend that you use the `apply` or `map_reduce` method of DataFrame to parallelize your serial traverse operations.

If you confirm that data traversing is necessary in your scenario and that the cost is acceptable, you can use the `to_pandas` method to convert your DataFrame to pandas DataFrame. You can also store DataFrame as a table and use `read_table` or Tunnel to read data.

12.Java sandbox limits

This topic describes the limits imposed by Java sandboxes on MaxCompute MapReduce and user-defined function (UDF) programs in distributed environments.

 **Note** The main programs of MapReduce jobs are not subject to these limits.

Limits:

- Direct access to local files is not allowed. You can access files only by using interfaces provided by MaxCompute MapReduce or MaxCompute Graph.
- Direct access to distributed file systems is not allowed. You can use only MaxCompute MapReduce or MaxCompute Graph to access tables.
- Java Native Interface (JNI) calls are not allowed.
- Java threads cannot be created, and Linux commands cannot be executed by sub-threads.
- Network access operations such as acquiring local IP addresses are not allowed.
- Java reflection limit: The suppressAccessChecks permission is prohibited. You cannot set a private attribute or method accessible to read private attributes or call private methods.

An access denied error is returned when you perform one of the preceding operations.

13. Volume lifecycle management

13.1. Overview

This topic provides an overview of the lifecycle of a MaxCompute volume.

In the previous versions of MaxCompute, a partition in a volume does not have a lifecycle and can exist indefinitely. Users must manage the lifecycles of volumes. In some cases, user management of volume lifecycles may cause a problem. For example, if the account used to delete a partition is different from the account used to create the partition, the delete operation fails. The new feature of volume lifecycle management in the current version solves this problem.

For more information about the volume lifecycle management feature, see the subsequent topic *Volume lifecycle operations*.

13.2. Manage the lifecycle of a volume

This topic describes how to manage the lifecycle of a MaxCompute volume.

Create a volume with a specified lifecycle

Example:

```
odps@ your_project>fs -mkv test_volume -lifecycle 7 "this is a test volume";
OK
```

Modify the lifecycle of a volume

Example:

```
odps@ your_project>fs -alter test_volume -lifecycle 3;
OK
```

View the lifecycle of a volume

Example:

```
odps@ your_project>fs -meta test_volume;
Comment: "this is a test volume"
Length: 0
File number: 0
Lifecycle: 3
OK
```

14. Spark on MaxCompute

14.1. Overview

This topic describes the definition and core features of Spark on MaxCompute.

Spark on MaxCompute is a solution developed by Alibaba Cloud to enable the seamless use of Spark on the MaxCompute platform. It supplements a wide variety of features to MaxCompute.

Spark on MaxCompute ensures the same user experience as native Spark and offers native Spark components and APIs. Spark on MaxCompute can access MaxCompute data sources and enhance security for multi-tenant scenarios. Spark on MaxCompute can also act as a management platform to share resources, storage, and user systems between Spark and MaxCompute jobs and ensure high performance at low costs. Spark can work with MaxCompute to create more efficient data processing solutions. Spark community applications can run seamlessly on Spark on MaxCompute.

Spark on MaxCompute has an independent data development node in DataWorks and supports data development in DataWorks.

14.2. Project resources

This topic describes the project resources that you must obtain before you can use Spark on MaxCompute.

Before you use Spark on MaxCompute, you must obtain the following project resources:

- Spark on MaxCompute release package: You must download the latest [Spark on MaxCompute](#) release package.
- Spark on MaxCompute plug-in: This is an open source plug-in. You can download it from [Aliyun Cupid SDK](#).

After you prepare the preceding project resources, you must complete environment configuration. Then, you can run related GitHub demos.

14.3. Environment settings

14.3.1. Decompress the Spark on MaxCompute release package

This topic describes the structure of the obtained folders and files after you decompress the Spark on MaxCompute release package.

Decompress the downloaded Spark on MaxCompute release package. The following code shows the structure of the obtained folders and files:

```
.
|-- R
|-- RELEASE
|-- __spark_libs__.zip
|-- bin
|-- conf
|-- cupid
|-- derby.log
|-- examples
|-- jars
|-- logs
|-- metastore_db
|-- python
|-- sbin
|-- yarn
```

14.3.2. Set environment variables

This topic describes how to set the environment variables required by Spark on MaxCompute.

Set environment variables based on your business requirements. The main environment variables are `JAVA_HOME` and `SPARK_HOME`.

Set `JAVA_HOME`

```
export JAVA_HOME=/path/to/jdk
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

Set `SPARK_HOME`

```
export SPARK_HOME=/path/to/spark_extracted_package
export PATH=$SPARK_HOME/bin:$PATH
```

If you use SparkR, install R in the `/home/admin/R` directory. Then, run the following command to set the path:

```
export PATH=/home/admin/R/bin/:$PATH
```

If you use PySpark, install Python 2.7. Then, run the following command to set the path:

```
export PATH=/path/to/python/bin/:$PATH
```

14.3.3. Configure `Spark-defaults.conf`

This topic describes how to configure the `Spark-defaults.conf` file, which is required when you use Spark on MaxCompute.

The `$SPARK_HOME/conf` directory contains a file named `Spark-defaults.conf`. Before you submit a Spark task to MaxCompute, you must configure your MaxCompute account in this file.

The following information is the default configuration in the file. You need only to specify the blank parameters based on your account information.

```
# OdpsAccount Info Setting
spark.hadoop.odps.project.name=
spark.hadoop.odps.access.id=
spark.hadoop.odps.access.key=
spark.hadoop.odps.end.point=
#spark.hadoop.odps.moye.trackurl.host=
#spark.hadoop.odps.cupid.webproxy.endpoint=
spark.sql.catalogImplementation=odps
# Spark-shell Setting
spark.driver.extraJavaOptions -Dscala.repl.reader=com.aliyun.odps.spark_repl.OdpsInteractiveReader -
Dscala.usejavacp=true
# SparkR Setting
# odps.cupid.spark.r.archive=/path/to/R-PreCompile-Package.zip
# Cupid Longtime Job
# spark.hadoop.odps.cupid.engine.running.type=longtime
# spark.hadoop.odps.cupid.job.capability.duration.hours=8640
# spark.hadoop.odps.moye.trackurl.dutation=8640
# spark.r.command=/home/admin/R/bin/Rscript
# spark.hadoop.odps.cupid.disk.driver.enable=false
spark.hadoop.odps.cupid.bearer.token.enable=false
spark.hadoop.odps.exec.dynamic.partition.mode=nonstrict
```

14.4. Quick start

This topic describes the basic operations for you to get started with Spark on MaxCompute.

1. Obtain and decompress the [Spark on MaxCompute](#) release package.
2. Set environment variables.

```
export SPARK_HOME=/path/to/spark-2.1.0-private-cloud-v3.1.0
export JAVA_HOME=/path/to/java/
```

3. Configure the spark-defaults.conf file.

```
cp $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-defaults.conf
```

Specify the blank parameters in `$SPARK_HOME/conf/spark-defaults.conf`.

```
# OdpsAccount Info Setting
spark.hadoop.odps.project.name=
spark.hadoop.odps.access.id=
spark.hadoop.odps.access.key=
spark.hadoop.odps.end.point=
#spark.hadoop.odps.moye.trackurl.host=
#spark.hadoop.odps.cupid.webproxy.endpoint=
spark.sql.catalogImplementation=odps
# spark-shell Setting
spark.driver.extraJavaOptions -Dscala.repl.reader=com.aliyun.odps.spark_repl.OdpsInteractiveReader -Dscala.usejavacp=true
# SparkR Setting
# odps.cupid.spark.r.archive=/path/to/R-PreCompile-Package.zip
# Cupid Longtime Job
# spark.hadoop.odps.cupid.engine.running.type=longtime
# spark.hadoop.odps.cupid.job.capability.duration.hours=8640
# spark.hadoop.odps.moye.trackurl.dutation=8640
# spark.r.command=/home/admin/R/bin/Rscript
# spark.hadoop.odps.cupid.disk.driver.enable=false
spark.hadoop.odps.cupid.bearer.token.enable=false
spark.hadoop.odps.exec.dynamic.partition.mode=nonstrict
```

4. Prepare spark-example.

```
git clone https://github.com/aliyun/aliyun-cupid-sdk.git
cd aliyun-cupid-sdk
mvn -T 1C clean install -DskipTests
```

5. Run SparkPi.

```
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class com.aliyun.odps.spark.examples.SparkPi /path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

If the following output is displayed, your operation is successful. Other logs may be included in the output.

```
18/02/09 15:52:28 INFO Client: Application report for application_1518162700322_1635034099 (state: FINISHED)
18/02/09 15:52:28 INFO Client:
  client token: N/A
  diagnostics: N/A
  ApplicationMaster host: ***
  ApplicationMaster RPC port: ***
  queue: queue
  start time: 1518162732343
  final status: SUCCEEDED
  tracking URL: http://***:80/proxyview/jobview/?h=http://***:80/api&p=odps_smoke_test&i=20180209075148695gbkp8e01&t=spark&id=application_1518162700322_1635034099&metaname=20180209075148695gbkp8e01&token=YkhjNXJWZ0dvdzVScXVFQWpCQWMra1RSZHVFPSPRFBTX09CTzoxMzY1OTM3MTUwNzcyMjEzLDE1MTg0MjE5MzUseyJTdGF0ZWl1bnQiOlt7IkFjdGlvb2I6I6WyJvZHBzO1JlYWQiXSwiRWZmZW50IjoieWxsb3ciLCJSZXNvdXJzZSI6WyJhY3M6b2RwczoqOnByb2p1Y3RzL29kcHNfc2lva2VfdGVzdC9pbmN0YW5jZXMvMjAxODAyMDkwNzUxNDg2OTVnYmtwOGUwMSJdfV0sIlZlcnNpb24iOiIiX3InO=
  user: user
18/02/09 15:52:28 INFO ShutdownHookManager: Shutdown hook called
18/02/09 15:52:28 INFO ShutdownHookManager: Deleting directory /tmp/spark-d77416ad-79a8-49f7-931d-0533663b5d85
```

14.5. Demo

This topic provides a demo on how to use Spark on MaxCompute.

The demo consists of the following steps:

1. Modify the configuration file.

Decompress the Spark package, go to the *conf* directory, and modify the following configuration items in the configuration file *spark-defaults.conf*:

```
spark.hadoop.odps.project.name=xxxx
spark.hadoop.odps.access.id=xxxx
spark.hadoop.odps.access.key=xxxx
spark.hadoop.odps.end.point=http://service.xxxx.xxxx.xxxxx.qd-inc.com:80/api
spark.hadoop.odps.cupid.distributedcache.mincopy=3
spark.hadoop.odps.cupid.distributedcache.maxcopy=3
spark.hadoop.odps.cupid.proxy.domain.name=jobview.xxxx.xxxx.xxxx.com
spark.hadoop.odps.cupid.history.server.address=10.xx.xx.xx:18080
```

Note

- You can obtain the first four configuration items from the *web_component.conf* file in the */cloud/app/odps-service-console/CupidFrontendServer#/cupid_web_proxy/current/conf.local/* directory.
- *spark.hadoop.odps.cupid.proxy.domain.name*: specifies the domain name of *odps_jobview_server_dns*. Remove *sparkui* at the beginning of the domain name.
- *spark.hadoop.odps.cupid.history.server.address*: specifies the IP address of the AG machine. Add the port number 18080 to the end of the IP address.

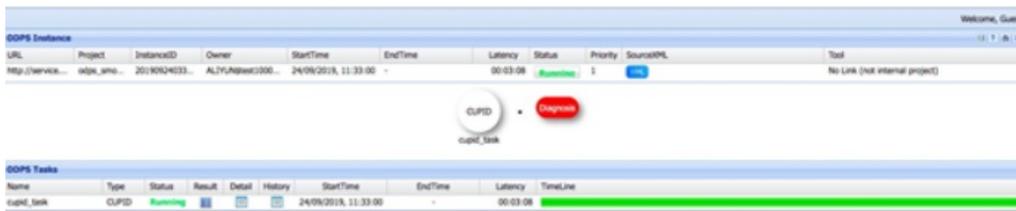
2. Start the program.

Go to the directory where the Spark package is decompressed and run the following commands:

```
bin/spark-submit
--class com.aliyun.odps.spark.examples.WordCount --master yarn-cluster
examples/spark-examples-2.0.0-SNAPSHOT-shaded.jar
```

3. Access the link in the command output.

Logview



SparkUI

Spark Jobs (7)

User: admin
 Total Uptime: 18 s
 Scheduling Mode: FIFO
 Completed Jobs: 3

Event Timeline

Completed Jobs (3)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	take at WordCount.scala:31	2019/09/24 11:33:14	41 ms	1/1 (1 skipped)	3/3 (10 skipped)
1	take at WordCount.scala:31	2019/09/24 11:33:14	77 ms	1/1 (1 skipped)	4/8 (10 skipped)
0	take at WordCount.scala:31	2019/09/24 11:33:13	0.8 s	2/2	1/1 (1)

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(3)	0	0.0 B / 1.2 GB	0.0 B	2	0	0	20	20	1 s (47 ms)	0.0 B	2.5 KB	4.6 KB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(3)	0	0.0 B / 1.2 GB	0.0 B	2	0	0	20	20	1 s (47 ms)	0.0 B	2.5 KB	4.6 KB

Executors

Show: 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.20.0.36:53360	Active	0	0.0 B / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stderr Thread stdout Dump	
1	a56a09211.cloud.a11.amtest43.24553	Active	0	0.0 B / 384.1 MB	0.0 B	1	0	0	11	11	0.6 s (24 ms)	0.0 B	556 B	3.2 KB	stderr Thread stdout Dump	
2	a56a09214.cloud.a11.amtest43.24557	Active	0	0.0 B / 384.1 MB	0.0 B	1	0	0	9	9	0.7 s (23 ms)	0.0 B	1.9 KB	1.4 KB	stderr Thread stdout Dump	

Showing 1 to 3 of 3 entries Previous 1 Next

Spark History

Spark Jobs (7)

User: admin
 Total Uptime:
 Scheduling Mode: FIFO
 Completed Jobs: 3

Event Timeline

Completed Jobs (3)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	take at WordCount.scala:31	2019/09/24 11:54:22	65 ms	1/1 (1 skipped)	3/3 (10 skipped)
1	take at WordCount.scala:31	2019/09/24 11:54:22	57 ms	1/1 (1 skipped)	4/8 (10 skipped)
0	take at WordCount.scala:31	2019/09/24 11:54:21	0.9 s	2/2	1/1 (1)

14.6. Common cases

14.6.1. WordCount example

This topic provides an example of using Spark on MaxCompute to run WordCount.

You must download the GitHub project and compile the project before you can run related demos.

```
git clone https://github.com/aliyun/aliyun-cupid-sdk.git
-- Download a GitHub project.
cd aliyun-cupid-sdk
mvn -T 1C clean install -DskipTests
-- Compile the GitHub project.
```

After you complete the preceding steps, JAR packages are generated. These JAR packages will be used to run the demos in this and the subsequent topics.

Example:

```
package com.aliyun.odps.spark.examples
import org.apache.spark.sql.SparkSession
object WordCount {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("WordCount")
      .getOrCreate()
    val sc = spark.sparkContext
    try {
      sc.parallelize(1 to 100, 10).map(word => (word, 1)).reduceByKey(_ + _, 10).take(100).foreach(p
rintln)
    } finally {
      sc.stop()
    }
  }
}
```

Run the following command to submit the job:

```
bin/spark-submit \
--master yarn-cluster \
--class com.aliyun.odps.spark.examples.WordCount \
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-
SNAPSHOT-shaded.jar
```

14.6.2. OSS access example

This topic provides an example of using Spark on MaxCompute to access OSS.

Example:

```
package com.aliyun.odps.spark.examples.oss
import org.apache.spark.sql.SparkSession
object SparkUnstructuredDataCompute {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("SparkUnstructuredDataCompute")
      .config("spark.hadoop.fs.oss.accessKeyId", "****")
      .config("spark.hadoop.fs.oss.accessKeySecret", "****")
      .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
      .getOrCreate()
    val sc = spark.sparkContext
    try {
      val pathIn = "oss://bucket/inputdata/"
      val inputData = sc.textFile(pathIn, 5)
      val cnt = inputData.count
      println(s"count: $cnt")
    } finally {
      sc.stop()
    }
  }
}
```

Run the following command to submit the job:

```
./bin/spark-submit
--jars cupid/hadoop-aliyun-package-3.0.0-alpha2-odps-jar-with-dependencies.jar
--class com.aliyun.odps.spark.examples.oss.SparkUnstructuredDataCompute
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

14.6.3. MaxCompute table read and write example

This topic provides an example of using Spark on MaxCompute to read data from and write data to a MaxCompute table and convert data to a Spark resilient distributed dataset (RDD).



Notice The project and table specified in the demo must exist or be changed to the specific project and table.

Example:

```
package com.aliyun.odps.spark.examples
import com.aliyun.odps.data.Record
import com.aliyun.odps.{ PartitionSpec, TableSchema}
import org.apache.spark.odps.OdpsOps
import org.apache.spark.sql.SparkSession
import scala.util.Random
object OdpsTableReadWrite {
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .appName("OdpsTableReadWrite")
      .getOrCreate()
    val sc = spark.sparkContext
    val projectName = sc.getConf.get("odps.project.name")
    try {
      val odpsOps = new OdpsOps(sc)
      // read from normal table via rdd api
      val rdd_0 = odpsOps.readTable(
        projectName,
        "cupid_wordcount",
        (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1))
      )
      //read from single partition column table via rdd api
      val rdd_1 = odpsOps.readTable(
        projectName,
        "dftest_single_parted",
        Array("pt=20160101"),
        (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("pt"))
      )
      // read from multi partition column table via rdd api
      val rdd_2 = odpsOps.readTable(
        projectName,
        "dftest_parted",
        Array("pt=20160101,hour=12"),
        (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("pt"), r.getString(3))
      )
      // read with multi partitionSpec definition via rdd api
      val rdd_3 = odpsOps.readTable(
```

```

    projectName,
    "cupid_partition_table1",
    Array("pt1=part1,pt2=part1", "pt1=part1,pt2=part2", "pt1=part2,pt2=part3"),
    (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("pt1"), r.g
etString("pt2"))
  )
  // save rdd into normal table
  val transfer_0 = (v: Tuple2[String, String], record: Record, schema: TableSchema) => {
    record.set("id", v._1)
    record.set(1, v._2)
  }
  odpsOps.saveToTable(projectName, "cupid_wordcount_empty", rdd_0, transfer_0, true)
  // save rdd into partition table with single partition spec
  val transfer_1 = (v: Tuple2[String, String], record: Record, schema: TableSchema) => {
    record.set("id", v._1)
    record.set("value", v._2)
  }
  odpsOps.saveToTable(projectName, "cupid_partition_table1", "pt1=test,pt2=dev", rdd_0, transfer
_1, true)
  // dynamic save rdd into partition table with multiple partition spec
  val transfer_2 = (v: Tuple2[String, String], record: Record, part: PartitionSpec, schema: Tabl
eSchema) => {
    record.set("id", v._1)
    record.set("value", v._2)
    val pt1_value = if (new Random().nextInt(10) % 2 == 0) "even" else "odd"
    val pt2_value = if (new Random().nextInt(10) % 2 == 0) "even" else "odd"
    part.set("pt1", pt1_value)
    part.set("pt2", pt2_value)
  }
  odpsOps.saveToTableForMultiPartition(projectName, "cupid_partition_table1", rdd_0, transfer_2,
true)
} catch {
  case ex: Exception => {
    throw ex
  }
} finally {
  sc.stop
}
}
}

```

Run the following command to submit the job:

```

bin/spark-submit \
--master yarn-cluster \
--class com.aliyun.odps.spark.examples.OdpsTableReadWrite \
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.j
ar

```

You can use one of the following methods to adjust the MaxCompute table read concurrency:

- Change the value of the `spark.hadoop.odps.input.split.size` parameter. A larger value indicates fewer Map tasks. The default value is 256 MB.
- Set `numPartition` in `OdpsOps.readTable`. The value determines the number of Map tasks. The number is calculated based on `spark.hadoop.odps.input.split.size`.

14.6.4. MaxCompute Table Spark-SQL example

This topic provides an example of using SQLContext in Spark on MaxCompute to read data from and write data to a MaxCompute Table.

Notice

- The project and table specified in the demo must exist or be changed to the specific project and table.
- Spark-defaults.conf must contain the setting `spark.sql.catalogImplementation = odps`.

Example:

```
package com.aliyun.odps.spark.examples
import org.apache.spark.sql.SparkSession
object OdpsTableReadWriteViaSQL {
  def main(args: Array[String]) {
    // please make sure spark.sql.catalogImplementation=odps in spark-defaults.conf
    // to enable odps catalog
    val spark = SparkSession
      .builder()
      .appName("OdpsTableReadWriteViaSQL")
      .getOrCreate()
    val projectName = spark.sparkContext.getConf.get("odps.project.name")
    val tableName = "cupid_wordcount"
    // get a ODPS table as a DataFrame
    val df = spark.table(tableName)
    println(s"df.count: ${df.count()}")
    // Just do some query
    spark.sql(s"select * from $tableName limit 10").show(10, 200)
    spark.sql(s"select id, count(id) from $tableName group by id").show(10, 200)
    // any table exists under project could be use
    // productRevenue
    spark.sql(
      """
      |SELECT product,
      |      category,
      |      revenue
      |FROM
      | (SELECT product,
      |      category,
      |      revenue,
      |      dense_rank() OVER (PARTITION BY category
      |                        ORDER BY revenue DESC) AS rank
      | FROM productRevenue) tmp
      |WHERE rank <= 2
      """stripMargin).show(10, 200)
    spark.stop()
  }
}
```

Run the following command to submit the job:

```
bin/spark-submit \
--master yarn-cluster \
--class com.aliyun.odps.spark.examples.OdpsTableReadWrite \
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

14.6.5. Example of the self-developed client mode

This topic provides an example of the self-developed client mode in Spark on MaxCompute.

For security purposes, machines in MaxCompute cannot be directly connected. Therefore, the yarn-client mode in native Spark cannot be used. To enable interaction, the MaxCompute team developed a proprietary client mode.

Example:

```
package com.aliyun.odps.spark.examples
import com.aliyun.odps.cupid.client.spark.client.CupidSparkClientRunner
object SparkClientNormalFT {
  def main(args: Array[String]) {
    val cupidSparkClient = CupidSparkClientRunner.getReadyCupidSparkClient()
    val jarPath = args(0) //client-jobexamples jar path
    val sparkClientNormalApp = new SparkClientNormalApp(cupidSparkClient)
    sparkClientNormalApp.runNormalJob(jarPath)
    cupidSparkClient.stopRemoteDriver()
  }
}
```

Run the following command to submit the job:

```
java -cp \
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar:
$SPARK_HOME/jars/* \
com.aliyun.odps.spark.examples.SparkClientNormalFT /path/to/aliyun-cupid-sdk/examples/client-jobexamples/target/client-jobexamples_2.11-1.0.0-SNAPSHOT.jar
```

14.6.6. MaxCompute Table PySpark example

This topic provides an example of using PySpark in Spark on MaxCompute to read data from and write data to a MaxCompute table.

Example:

```
from odps.odps_sdk import OdpsOps
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext, DataFrame
if __name__ == '__main__':
    conf = SparkConf().setAppName("odps_pyspark")
    sc = SparkContext(conf=conf)
    sql_context = SQLContext(sc)
    project_name = "cupid_testal"
    in_table_name = "cupid_wordcount"
    out_table_name = "cupid_wordcount_py"
    normal_df = OdpsOps.read_odps_table(sql_context, project_name, in_table_name)
    for i in normal_df.sample(False, 0.01).collect():
        print i
    print "Read normal odps table finished"
    OdpsOps.write_odps_table(sql_context, normal_df.sample(False, 0.001), project_name, out_table_name)
    print "Write normal odps table finished"
```

Run the following command to submit the job:

```
spark-submit \
--master yarn-cluster \
--jars /path/to/aliyun-cupid-sdk/external/cupid-datasource/target/cupid-datasource_2.11-1.0.0-SNAPSHOT.jar \
--py-files /path/to/aliyun-cupid-sdk/examples/spark-examples/src/main/python/odps.zip \
/path/to/aliyun-cupid-sdk/examples/spark-examples/src/main/python/odps_table_rw.py
```

14.6.7. MLlib example

This topic provides an example of using MLlib in Spark on MaxCompute.

We recommend that you use OSS for read and write operations in the MLlib model.

Example:

```
package com.aliyun.odps.spark.examples.mllib
import org.apache.spark.mllib.clustering.KMeans._
import org.apache.spark.mllib.clustering.{ KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.sql.SparkSession
object KmeansModelSaveToOss {
  val modelOssDir = "oss://bucket/kmeans-model"
  def main(args: Array[String]) {
    //1. train and save the model
    val spark = SparkSession
      .builder()
      .appName("KmeansModelSaveToOss")
      .config("spark.hadoop.fs.oss.accessKeyId", "****")
      .config("spark.hadoop.fs.oss.accessKeySecret", "****")
      .config("spark.hadoop.fs.oss.endpoint", "****")
      .getOrCreate()
    val sc = spark.sparkContext
    val points = Seq(
      Vectors.dense(0.0, 0.0),
      Vectors.dense(0.0, 0.1),
      Vectors.dense(0.1, 0.0),
      Vectors.dense(9.0, 0.0),
      Vectors.dense(9.0, 0.2),
      Vectors.dense(9.2, 0.0)
    )
    val rdd = sc.parallelize(points, 3)
    val initMode = K_MEANS_PARALLEL
    val model = KMeans.train(rdd, k = 2, maxIterations = 2, runs = 1, initMode)
    val predictResult1 = rdd.map(feature => "cluster id: " + model.predict(feature) + " feature:" +
      feature.toArray.mkString(",")).collect
    println("modelOssDir=" + modelOssDir)
    model.save(sc, modelOssDir)
    //2. predict from the oss model
    val modelLoadOss = KMeansModel.load(sc, modelOssDir)
    val predictResult2 = rdd.map(feature => "cluster id: " + modelLoadOss.predict(feature) + " feature:" +
      feature.toArray.mkString(",")).collect
    assert(predictResult1.size == predictResult2.size)
    predictResult2.foreach(result2 => assert(predictResult1.contains(result2)))
  }
}
```

Run the following command to submit the job:

```
./bin/spark-submit
--jars cupid/hadoop-aliyun-package-3.0.0-alpha2-odps-jar-with-dependencies.jar
--class com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

14.6.8. PySpark interactive execution example

This topic provides an example of PySpark interactive execution in Spark on MaxCompute.

PySpark can run only on a host that can be directly connected to a computing cluster.

Install Python 2.7 and set a path:

```
export PATH=/path/to/python/bin/:$PATH
```

Run the following start command:

```
bin/pyspark --master yarn-client
```

Perform interactive execution:

```
df=spark.sql("select * from spark_user_data")
df.show()
```

14.6.9. Spark-shell interactive execution example (read tables)

This topic provides an example of Spark-shell interactive execution (read tables) in Spark on MaxCompute.

Spark-shell can run only on a host that can be directly connected to a computing cluster.

Run the following start command:

```
bin/spark-shell --master yarn
```

Perform interactive execution:

```
sc.parallelize(0 to 100, 2).collect
sql("show tables").show
sql("select * from spark_user_data").show(200,100)
```

14.6.10. Spark-shell interactive execution example (Spark MLib and OSS read/write)

This topic provides an example of Spark-shell interactive execution (Spark MLib and OSS read/write) in Spark on MaxCompute.

Spark-shell can run only on a host that can be directly connected to a computing cluster.

Add the following configuration to conf/spark-defaults.conf:

```
spark.hadoop.fs.oss.accessKeyId=***
spark.hadoop.fs.oss.accessKeySecret=***
spark.hadoop.fs.oss.endpoint=***
```

Run the following start command:

```
bin/spark-shell --master yarn --jars cupid/hadoop-aliyun-package-3.0.0-alpha2-odps-jar-with-dependencies.jar
```

Perform interactive execution:

```
import org.apache.spark.mllib.clustering.KMeans._
import org.apache.spark.mllib.clustering.{ KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors
val modelOssDir = "oss://your_bucket/kmeans-model"
val points = Seq(
  Vectors.dense(0.0, 0.0),
  Vectors.dense(0.0, 0.1),
  Vectors.dense(0.1, 0.0),
  Vectors.dense(9.0, 0.0),
  Vectors.dense(9.0, 0.2),
  Vectors.dense(9.2, 0.0)
)
val rdd = sc.parallelize(points, 3)
val initMode = K_MEANS_PARALLEL
val model = KMeans.train(rdd, k = 2, maxIterations = 2, runs = 1, initMode)
val predictResult1 = rdd.map(feature => "cluster id: " + model.predict(feature) + " feature:" + feature.toArray.mkString(",")).collect
println("modelOssDir=" + modelOssDir)
model.save(sc, modelOssDir)
val modelLoadOss = KMeansModel.load(sc, modelOssDir)
val predictResult2 = rdd.map(feature => "cluster id: " + modelLoadOss.predict(feature) + " feature:" + feature.toArray.mkString(",")).collect
assert(predictResult1.size == predictResult2.size)
predictResult2.foreach(result2 => assert(predictResult1.contains(result2)))
```

14.6.11. Example of SparkR interactive execution

This topic provides an example of the SparkR interactive execution in Spark on MaxCompute.

SparkR can be run only on a machine that can directly connect to a computing cluster. In addition, you must install R in the `/home/admin/R` directory and specify the path:

```
export PATH=/home/admin/R/bin/:$PATH
```

The following command is used to start SparkR.

```
bin/sparkR --master yarn --archives ./R/R.zip
```

The following commands are run in interactive mode.

```
df <- as.DataFrame(faithful)
df
head(select(df, df$eruptions))
head(select(df, "eruptions"))
head(filter(df, df$waiting < 50))
results <- sql("FROM spark_user_data SELECT *")
head(results)
```

14.6.12. Example of PageRank with Apache Spark GraphX

This topic provides an example of PageRank with Apache Spark GraphX in Spark on MaxCompute.

Spark on MaxCompute supports Apache Spark GraphX.

Example:

```
package com.aliyun.odps.spark.examples.graphx
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
object PageRank {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("pagerank")
    val sc = new SparkContext(conf)
    // construct vertices
    val users: RDD[(VertexId, Array[String])] = sc.parallelize(List(
      "1,BarackObama,Barack Obama",
      "2,ladygaga,Goddess of Love",
      "3,jeresig,John Resig",
      "4,justinbieber,Justin Bieber",
      "6,matei_zaharia,Matei Zaharia",
      "7,odersky,Martin Odersky",
      "8,anonsys"
    )).map(line => line.split(",")).map(parts => (parts.head.toLong, parts.tail))
    // construct edges
    val followers: RDD[Edge[Double]] = sc.parallelize(Array(
      Edge(2L,1L,1.0),
      Edge(4L,1L,1.0),
      Edge(1L,2L,1.0),
      Edge(6L,3L,1.0),
      Edge(7L,3L,1.0),
      Edge(7L,6L,1.0),
      Edge(6L,7L,1.0),
      Edge(3L,7L,1.0)
    ))
    // construct graph
    val followerGraph: Graph[Array[String], Double] = Graph(users, followers)
    // restrict the graph to users with usernames and names
    val subgraph = followerGraph.subgraph(vpred = (vid, attr) => attr.size == 2)
    // compute PageRank
    val pageRankGraph = subgraph.pageRank(0.001)
    // get attributes of the top pagerank users
    val userInfoWithPageRank = subgraph.outerJoinVertices(pageRankGraph.vertices) {
      case (uid, attrList, Some(pr)) => (pr, attrList.toList)
      case (uid, attrList, None) => (0.0, attrList.toList)
    }
    println(userInfoWithPageRank.vertices.top(5)(Ordering.by(_._2._1)).mkString("\n"))
  }
}
```

Run the following command to submit a job:

```
bin/spark-submit \
--master yarn-cluster \
--class com.aliyun.odps.spark.examples.graphx.PageRank \
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

14.6.13. Example of Spark Streaming- NetworkWordCount

This topic provides an example of Spark Streaming-`NetworkWordCount` in Spark on MaxCompute.

Native Spark Streaming is supported. To use `NetworkWordCount`, you must install netcat on your on-premises machine, and then run the following command:

```
$ nc -lk 9999
```

The input in the command terminal becomes the input of Spark Streaming.

Example:

```
package com.aliyun.odps.spark.examples.streaming
import org.apache.spark.SparkConf
import org.apache.spark.examples.streaming.StreamingExamples
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.{Seconds, StreamingContext}
object NetworkWordCount {
  def main(args: Array[String]) {
    if (args.length < 2) {
      System.err.println("Usage: NetworkWordCount <hostname> <port>")
      System.exit(1)
    }
    StreamingExamples.setStreamingLogLevels()
    // Create the context with a 1 second batch size
    val sparkConf = new SparkConf().setAppName("NetworkWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(1))
    // Create a socket stream on target ip:port and count the
    // words in input stream of \n delimited text (eg. generated by 'nc')
    // Note that no duplication in storage level only for running locally.
    // Replication necessary in distributed scenario for fault tolerance.
    val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY_AND_DISK_SER)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

Run the following command to submit a job:

```
bin/spark-submit \
--master local[4] \
--class com.aliyun.odps.spark.examples.streaming.NetworkWordCount \
/path/to/aliyun-cupid-sdk/examples/spark-examples/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar localhost 9999
```

14.7. Maven dependencies

This topic describes Maven dependencies of Spark on MaxCompute.

The GitHub project described in the Common cases topic can be used as a template for your quick start. For custom development, use the following pom.xml file.

 **Note** Make sure that the Spark community edition is 2.3.0 and the scope is provided.

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.11</artifactId>
  <version>2.3.0</version>
  <scope>provided</scope>
</dependency>
```

The MaxCompute plug-in has been released to the Maven repository. You must add the following dependencies to the pom.xml file:

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-core_2.11</artifactId>
  <version>1.0.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-client_2.11</artifactId>
  <version>1.0.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-datasource_2.11</artifactId>
  <version>1.0.0</version>
</dependency>
```

Dependencies in the Maven repository:

- **Core code of the Cupid platform:** encapsulates the APIs that are used to submit Cupid tasks and the APIs that parent and child processes use to read data from and write data to tables.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-core_2.11</artifactId>
  <version>1.0.0</version>
</dependency>
```

- **Datasource:** encapsulates Spark-related APIs that are used to read data from and write data to MaxCompute tables.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-datasource_2.11</artifactId>
  <version>1.0.0</version>
```

- **Client:** encapsulates an SDK in Cupid client mode.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-client_2.11</artifactId>
  <version>1.0.0</version>
</dependency>
```

14.8. Special notes

14.8.1. Running mode

This topic describes the running mode of Spark on MaxCompute.

Spark on MaxCompute supports three running modes: local, cluster, and DataWorks.

Local mode

The local mode facilitates code debugging for applications. In local mode, you can use Spark on MaxCompute the same way as that described in the Apache Spark community. You can also use Tunnel to read and write data from and to MaxCompute tables. In this mode, you can use either an integrated development environment (IDE) or the command line to run Spark on MaxCompute.

If Spark on MaxCompute works in this mode, you must add the configuration `spark.master=local[N]`. `N` indicates the CPU cores required to implement this mode.

In local mode, table read and write operations are implemented by using Tunnel. Therefore, you must add a Tunnel configuration item to the `spark-defaults.conf` file.

 **Note** Enter the endpoint based on the region and network environment in which your MaxCompute project resides.

The following code provides an example on how to use the command line to run Spark on MaxCompute in local mode:

```
1.bin/spark-submit --master local[4] \  
--class com.aliyun.odps.spark.examples.SparkPi \  
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-version-shaded.jar
```

Cluster mode

In cluster mode, you must specify the Main method as the entry point of a custom application. A Spark job ends when Main succeeds or fails. This mode is suitable for offline jobs. You can use Spark on MaxCompute in this mode with DataWorks to schedule jobs.

The following code provides an example on how to use the command line to run Spark on MaxCompute in cluster mode:

```
1.bin/spark-submit --master yarn-cluster \  
-class SparkPi \  
${ProjectRoot}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-version-shaded.jar
```

DataWorks mode

You can run offline jobs of Spark on MaxCompute that is in cluster mode in DataWorks to integrate and schedule other types of nodes.

Procedure:

1. Upload the resources in the DataWorks business flow and click **Submit**.
2. In the created business flow, select **ODPS Spark** from **Data Analytics**.
3. Double-click the Spark node and define the Spark job.

Select a Spark version, a development language, and a resource file for the job. The resource file is the JAR file that is uploaded and published in the business flow.

You can specify configuration items, such as the number of executors and the memory size, for the job that you want to submit.

You must also specify the endpoint configuration item `spark.hadoop.odps.cupid.webproxy.endpoint` of Spark on MaxCompute.

 **Note** Enter the endpoint of the region in which your MaxCompute project resides, for example, <http://service.cn.maxcompute.aliyun-inc.com/api>.

4. You can manually run the Spark node to view the task log and obtain the URLs of Logview and Jobview from the log for further analysis and diagnosis.
5. After the Spark job is defined, orchestrate and schedule services of different types in the business flow as required.

14.8.2. Spark Streaming tasks

This topic describes the configurations that are required to run Spark Streaming tasks.

MaxCompute supports Spark Streaming. To support Spark Streaming tasks that run for a long period of time, you must add the following special configurations to the `spark-defaults.conf` file:

```
spark.hadoop.odps.cupid.engine.running.type=longtime
# Set the running type of a task to longtime so that the task is not reclaimed.
spark.hadoop.odps.cupid.job.capability.duration.hours=25920
# Specify the running duration.
spark.yarn.maxAppAttempts=10
# Specify the maximum number of retries for a failover.
spark.streaming.receiver.writeAheadLog.enable=true
# Determine whether to enable the write-ahead logging mode. This feature prevents data loss but reduces data processing efficiency.
```

14.8.3. Job diagnosis

This topic describes job diagnosis of Spark on MaxCompute.

After you submit a job, you must check the job log to determine whether the job is submitted and executed as expected. MaxCompute provides Logview and Spark web UI for you to diagnose jobs.

The following example demonstrates how to submit a job in `spark-submit` mode. Logs are also generated when you use DataWorks to run Spark jobs.

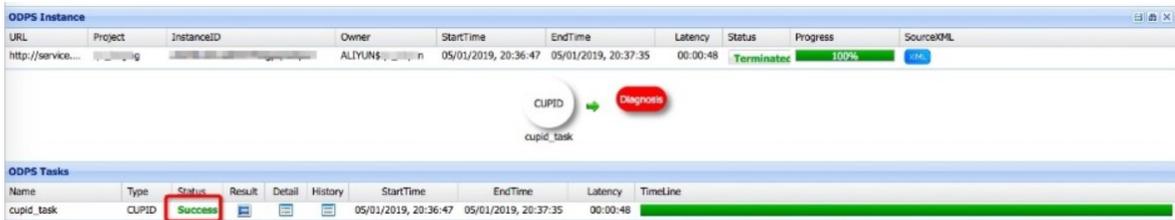
```
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class SparkPi /tmp/spark-2.x-demo/target/Alispark-2.x-quickstart-1.0-SNAPSHOT-shaded.jar
```

After the job is submitted, MaxCompute creates an instance and displays the Logview URL of the instance in the log.

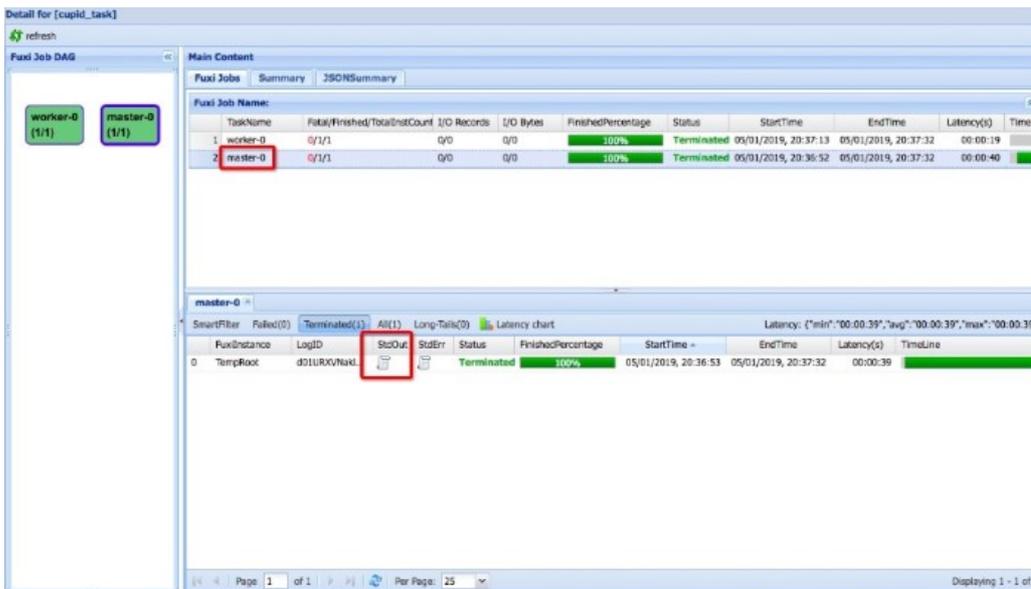
```
19/01/05 20:36:47 INFO YarnClientImplUtil: logview url: http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/api&p=qn_beijing&i=xxx&token=xxx
-- <The operation succeeds if an output similar to the following result is displayed.>
19/01/05 20:37:34 INFO Client:
  client token: N/A
  diagnostics: N/A
  ApplicationMaster host: 11.220.xxx.xxx
  ApplicationMaster RPC port: 30002
  queue: queue
  start time: 1546691807945
  final status: SUCCEEDED
  tracking URL: http://jobview.odps.aliyun.com/proxyview/jobview/?h=http://service.cn.maxcompute.aliyun-inc.com/api&p=project_name&i=xxx&t=spark&id=application_xxx&metaname=xxx&token=xxx
```

Use Logview to diagnose a job

1. View basic execution information about the task of the CUPID type in a browser based on the Logview URL.



2. Click the progress bar of the task whose TaskName is master-0. In the lower pane, click All and find TempRoot in the FuxiInstance column.



3. Click the icon in the StdOut column that corresponds to TempRoot to view the output of SparkPi.



Use Spark web UI to diagnose a job

The tracking URL in the log indicates that your job is submitted to the MaxCompute cluster. This URL is crucial because it is the URL of both Spark web UI and History Server.

1. Access the URL in a browser to track the running status of your Spark job.

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active[2]	0	0.0 B / 5.3 GB	0.0 B	2	0	0	2	2	2 s (0.1 s)	0.0 B	0.0 B	0.0 B
Dead[0]	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total[2]	0	0.0 B / 5.3 GB	0.0 B	2	0	0	2	2	2 s (0.1 s)	0.0 B	0.0 B	0.0 B

Executors

Show 20 entries

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	cupid-11-220-203-36-45885	Active	0	0.0 B / 2.1 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stderr stdout
1	worker50dcb649-ccc8-4151-a605-0b2034658fa3cupid-11-220-216-77-43705	Active	0	0.0 B / 3.2 GB	0.0 B	2	0	0	2	2	2 s (0.1 s)	0.0 B	0.0 B	0.0 B	stderr stdout

Showing 1 to 2 of 2 entries

2. Click **StdOut** in the Logs column that corresponds to the driver to view the output of the Spark job.

jobview.odps.aliyun.com/logsview/...

Pi is roughly 3.1434

14.9. APIs supported by Spark

14.9.1. Spark Shell

This topic describes the Spark shell API of Spark on MaxCompute.

Run the following commands to start the application of the Spark shell API:

```
$cd $SPARK_HOME
-- Go to the Spark shell directory.
$bin/spark-shell --master yarn
-- Specify the running mode and start the application.
```

Example:

```
sc.parallelize(0 to 100, 2).collect
sql("show tables").show
sql("select * from spark_user_data").show(200,100)
```

14.9.2. Spark R

This topic describes the SparkR API of Spark on MaxCompute.

Run the following commands to start the application of the SparkR API:

```
$mkdir -p /home/admin/R && unzip ./R/R.zip -d /home/admin/R/
# Create the R directory and decompress the R.zip package in this directory.
$export PATH=/home/admin/R/bin/:$PATH
-- Configure environment variables.
$bin/sparkR --master yarn --archives ./R/R.zip
-- Specify the running mode and start the application.
```

Example:

```
df <- as.DataFrame(faithful)
df
head(select(df, df$eruptions))
head(select(df, "eruptions"))
head(filter(df, df$waiting < 50))
results <- sql("FROM spark_user_data SELECT *")
head(results)
```

14.9.3. Spark SQL

This topic describes the Spark SQL API of Spark on MaxCompute.

Run the following commands to start the application of the Spark SQL API:

```
$cd $SPARK_HOME
-- Go to the Spark SQL directory.
$bin/spark-sql --master yarn
-- Specify the running mode and start the application.
```

Example:

```
show tables;
select * from spark_user_data limit 3;;
quit;
```

14.9.4. Spark JDBC

This topic describes the Spark Java Database Connectivity (JDBC) API of Spark on MaxCompute.

Run the following commands to start the application of the Spark JDBC API:

```
$sbin/stop-thriftserver.sh
-- Stop a thread.
/sbin/start-thriftserver.sh
-- Restart a thread.
$bin/beeline
-- Start the application.
```

Example:

```
!connect jdbc:hive2://localhost:10000/odps_smoke_test
show tables;
select * from mr_input limit 3;
!quit
```

14.10. Dynamic resource allocation of Spark

This topic describes the dynamic resource allocation (DRA) feature of Spark on MaxCompute. This feature is supported only in Spark 2.4.5 or later, which corresponds to Apsara Stack Enterprise Edition V3.13.0 or later.

Background information

Spark provides a large number of parameters to implement a wide range of semantics. You can use the default values for most parameters. However, specific parameters require manual configurations. The spark.executor.instances parameter is the most complex parameter to configure.

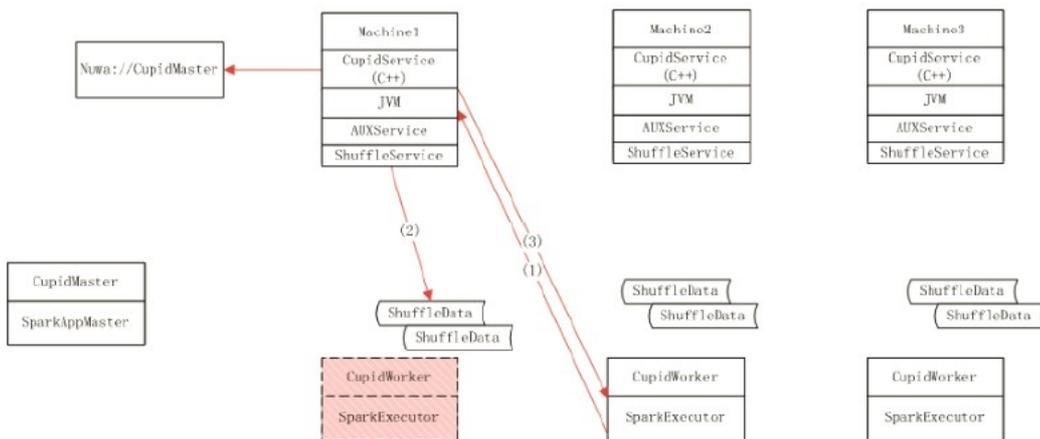
If this parameter is set to an excessively small value, operations may run slowly or fail due to an out of memory (OOM) error or lack of disk space. If this parameter is set to an excessively large value, resources are wasted.

Even the optimal value that is obtained based on full understanding of the data and logic is not reliable. For complex jobs, the required number of executors at different stages is different, and different resources are required when a job is running. Fixed configurations of resources cause waste. If long tail latency occurs, idle resources are occupied by other executors even for simple jobs.

Solution

The best solution is to allocate resources based on your business requirements. DRA is a solution that addresses this issue. CupidService developed by the Cupid team supports native DRA. The following figure shows the implementation process of DRA.

Note For more information about native DRA in the Apache Spark community, see [Spark help](#) and [Spark configuration](#).



Enable DRA

You can add the following configurations to enable DRA without the need to modify the code:

```

spark.hadoop.odps.cupid.shuffleservice.enable=true // Required. This parameter specifies whether to
enable the shuffle service in CupidService.
spark.hadoop.odps.cupid.disk.driver.enable=true // Required. This parameter is a dependent item.
spark.dynamicAllocation.enabled=true // Required. This parameter specifies whether to enable DRA in
Spark.
spark.shuffle.service.enabled=true // Required. This parameter is a dependent item.
spark.shuffle.service.port=7338 // Required. This parameter indicates the port used for the shuffle
service.
spark.authenticate=true // Required. This parameter specifies whether to enable authentication.
spark.dynamicAllocation.maxExecutors=128 // Optional. This parameter indicates the maximum number of
executors.
spark.dynamicAllocation.minExecutors=1 // Optional. This parameter indicates the minimum number of e
xecutors.
spark.dynamicAllocation.initialExecutors=1 // Optional. This parameter indicates the initial number
of executors.
spark.dynamicAllocation.executorIdleTimeout=60s // Optional. This parameter indicates the waiting pe
riod before idle executors are released.
    
```

If DRA is enabled, the `spark.executor.instances` parameter is optional. If you configure the `spark.executor.instances` parameter, the effect of this parameter is equivalent to the effect of the `spark.dynamicAllocation.initialExecutors` parameter.

14.11. FAQ of Spark on MaxCompute

This topic provides answers to some frequently asked questions about Spark on MaxCompute.

How do I migrate code of the open source Spark to Spark on MaxCompute?

The migration method depends on whether your job needs to access MaxCompute tables or Object Storage Service (OSS):

- If the job does not need to access MaxCompute tables or OSS, run your JAR package directly. Note that you must set the dependency of Spark or Hadoop to provided.
- If the job needs to access MaxCompute tables, you need only to configure related dependencies and repackage them.
- If the job needs to access OSS, you need only to configure related dependencies and repackage them.

How do I use Spark on MaxCompute to access services in a VPC?

Spark on MaxCompute does not allow access to services in a VPC. If you want to access a service in a VPC, submit a ticket to contact the MaxCompute technical support team.

The following error message appears, which indicates that the ID and key in the `spark-defaults.conf` file are invalid. What do I do?

Error:

```
Stack:  
com.aliyun.odps.OdpsException: ODPS-0410042:  
Invalid signature value - User signature dose not match
```

Check whether the ID and key in the `spark-defaults.conf` file are consistent with the AccessKey ID and AccessKey secret that you obtain from the Apsara Uni-manager Management Console.

The following error message appears, which indicates that I do not have permissions. What do I do?

Error:

```
Stack:  
com.aliyun.odps.OdpsException: ODPS-0420095:  
Access Denied - Authorization Failed [4019], You have NO privilege 'odps:CreateResource' on {acs:odps:*:projects/*}
```

The project owner must grant the READ and CREATE permissions on resources to you.

The following error message appears, which indicates that the project does not support Spark jobs. What do I do?

Error:

```
Exception in thread "main" org.apache.hadoop.yarn.exceptions.YarnException: com.aliyun.odps.OdpsException: ODPS-0420095: Access Denied - The task is not in release range: CUPID
```

Check whether the Spark on MaxCompute service is provided in the region where the project resides. In addition, check whether the configurations in the `spark-defaults.conf` file are consistent with those described in the MaxCompute documentation. If the Spark on MaxCompute service is provided in the region and the configurations in the `spark-defaults.conf` file are correct, submit a ticket.

When a task is running, the following error message appears, which indicates that local storage space is insufficient. What do I do?

Error:

```
No space left on device
```

Spark on MaxCompute uses online storage to replace local storage. Shuffled data and overflow data of BlockManager are all stored online. Therefore, you must check the setting of the online storage space. The storage space is determined by the `spark.hadoop.odps.cupid.disk.driver.device_size` parameter. The default storage space is 20 GB and the maximum storage space is 100 GB. If the error persists after you increase the storage space to 100 GB, conduct further analysis.

The most common cause is data skew. Data is unevenly distributed among blocks during the shuffle and cache processes. If this is the case, you can decrease the number of concurrent tasks in each executor (`spark.executor.cores`) or increase the number of executors (`spark.executor.instances`).

15. Elasticsearch on Maxcompute

15.1. Overview

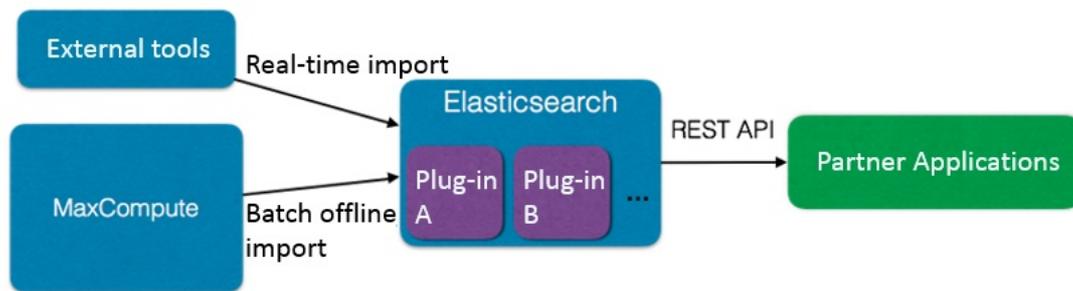
This topic provides an overview of Elasticsearch on MaxCompute and its core features.

Elasticsearch on MaxCompute is an enterprise-class full-text index system developed by Alibaba Cloud to retrieve large amounts of data. It provides near-real-time (NRT) search for government agencies and enterprises. It has the following benefits:

- Provides scalable full-text index services and supports native Elasticsearch API operations.
- Supports data import from multiple heterogeneous data sources based on the API development.
- Supports cluster- and business-level O&M management.

As a combination of Elasticsearch and MaxCompute, Elasticsearch on MaxCompute provides efficient, core massive data search engine services by leveraging unified scheduling and management of MaxCompute. In addition, Elasticsearch on MaxCompute can be used with the plug-ins of open source Elasticsearch to provide a wide range of index features.

Elasticsearch on MaxCompute allows you to use tools to import data from external sources in real time. You can also import offline data from MaxCompute. After the imported data is indexed, Elasticsearch on MaxCompute provides index services by using RESTful APIs. The following figure shows the usage of Elasticsearch on MaxCompute.



15.2. Workflow

15.2.1. Overview

This topic describes the workflow of Elasticsearch on MaxCompute.

Elasticsearch on MaxCompute is developed based on the open source Elasticsearch. It can run the Elasticsearch service on MaxCompute clusters.

In the MaxCompute client, you can start and manage your Elasticsearch service as required, including the number of nodes, disk space, memory size, and custom settings. The resources consumed by Elasticsearch are counted towards your MaxCompute quota.

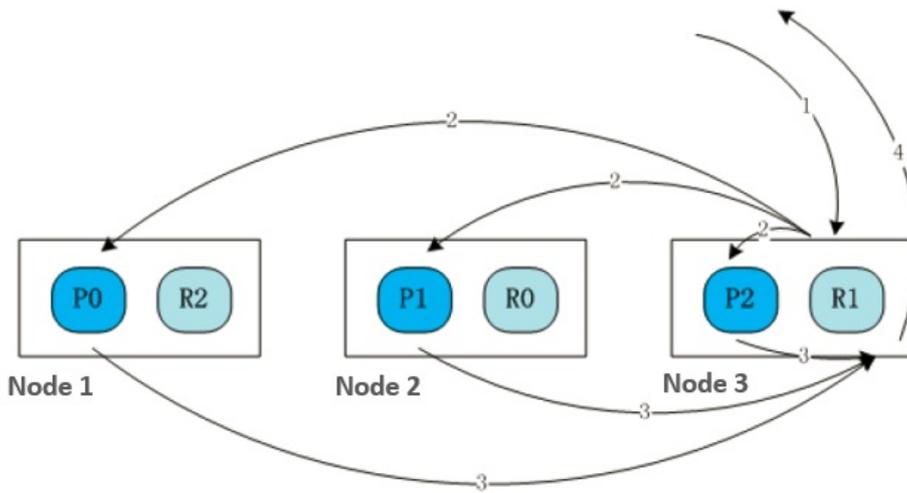
For more information about the process for starting Elasticsearch, see [Elasticsearch typical practice](#).

The following topics describe the workflows of the features of Elasticsearch on MaxCompute.

15.2.2. Distributed search workflow

This topic describes the distributed search workflow of Elasticsearch on MaxCompute.

The following figure shows the distributed search workflow.



In the preceding figure, the cluster consists of three nodes. The index has three shards: P0, P1, and P2. These shards are distributed across the three nodes. Each shard is replicated in 1:1 mode. Three replicas are generated: R0, R1, and R2.

1. A user sends a search request to Node 3.
2. After Node 3 receives the request, it sends a search request (2) to P0, P1, and P2 based on the recorded index shard information.
3. The nodes in which P0, P1, and P2 are located search for the requested information in the specified shards. A search result message (3) is sent to Node 3.
4. Node 3 collects the search results from other nodes and returns the search results to the user in an acknowledgment message (4).

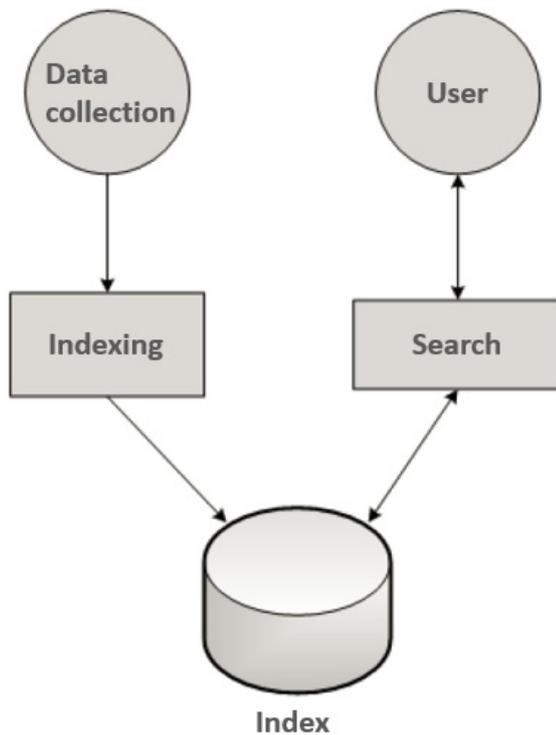
Note

- The search speed is increased because multiple nodes perform the search at the same time.
- The performance of distributed search improves with the increase of nodes.

15.2.3. Full-text index workflow

This topic describes the full-text index workflow of Elasticsearch on MaxCompute.

The following figure shows the full-text index workflow.



1. The data collection module collects both structured and unstructured data, converts the data into the field+value format, and submits the data to the indexing module.
2. After the indexing module receives the field+value data, it tokenizes and creates an inverted index based on the predefined indexing method of the field and saves the data to the index. The type, indexing method, and tokenization method of each field are configured on the retrieval management platform.
3. A user sends a search request. After the search processing module receives and processes the request, the search index, field, and query statement are obtained. The specified record list is included in the inverted index data.
4. The indexing module returns data that meets the requirements such as sorting rules and the number of requests.

15.2.4. Authentication process

This topic describes the authentication process of Elasticsearch on MaxCompute.

Authentication process:

1. Elasticsearch on MaxCompute allows you to perform retrieval management and O&M on MaxCompute. To perform these operations, you must log on to the retrieval management or O&M platform that is provided by Elasticsearch on MaxCompute. During logon, you are redirected to the authentication module for authentication. If the authentication fails, you are not allowed to access the related platform.
2. The administrator can use the MaxCompute client to add Elasticsearch users and grant permissions for the users.
3. The system authenticates all users who attempt to access index libraries. After you pass the authentication, you are allowed to retrieve data or perform operations on data in the libraries.

15.3. Quick start

This topic describes how to use Elasticsearch on MaxCompute. It guides you through the basic use of Elasticsearch on MaxCompute.

Before you start an Elasticsearch cluster, make sure that you have determined the following information:

- **Node planning:** Determine the number of nodes that are required for each role in an Elasticsearch cluster. By default, an Elasticsearch cluster has two roles, master and data. Each role is deployed on three nodes. You can add nodes to the cluster at any time.
- **Resource planning:** Determine the vCPU, memory, and disk space resources that are required for each node. The resources configured for each node cannot be changed. By default, each node is assigned 8 GB of memory and 20 GB of disk space.

 **Note** Only 50% of the memory allocated to a data node is used for the JVM heap.

- **Elasticsearch configuration:** Determine the running configurations of nodes in an Elasticsearch cluster, such as the queue size of bulk requests, and support for cross-domain HTTP requests.

After you determine the preceding information, you can start your Elasticsearch cluster in the MaxCompute client.

The following example demonstrates how to quickly start a small Elasticsearch cluster based on the default configuration. In the following example, the name of the Elasticsearch cluster you want to start is `es_first_cluster`.

1. Download the [MaxCompute client](#) that supports Elasticsearch. Configure the AccessKey pair, project, and endpoint.
2. Run `odpscmd` and run the following command to start the Elasticsearch cluster:

```
server create es_first_cluster type elasticsearch_mdu;
```

Wait for several minutes. If OK is returned, the Elasticsearch cluster is started. You must create an Elasticsearch user to access the cluster.

3. In `odpscmd`, create an Elasticsearch user with the `ALL_ACCESS` permission.

```
server execute es_first_cluster create user admin with password 123456|all_access;
```

If OK is returned, the Elasticsearch user is created.

4. Access the started Elasticsearch cluster. In the following example, the MaxCompute project that is used to access the Elasticsearch cluster is `prj1`. Run the following command to return information about the Elasticsearch cluster:

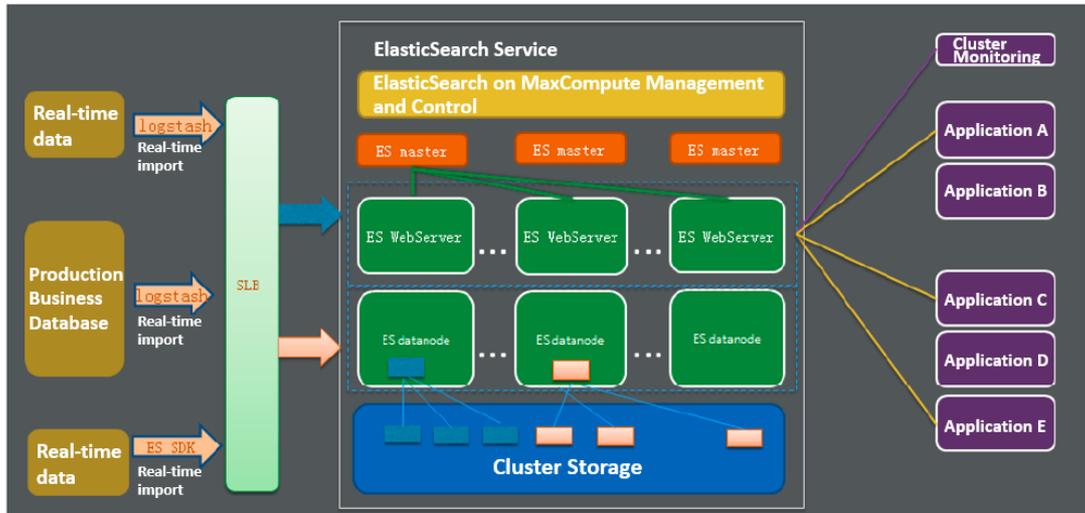
```
curl -u admin:123456 http://search.aliyun.com:9200/prj1.es_first_cluster;
```

 **Notice** To delete the Elasticsearch cluster, run the `server delete es_first_cluster` command. If you run this command, the Elasticsearch cluster is **permanently deleted, and data cannot be restored**. We recommend that you proceed with caution.

15.4. Support for Elasticsearch applications

15.4.1. Typical practice of Elasticsearch on MaxCompute

This topic describes the typical practice of Elasticsearch on MaxCompute.



Elasticsearch on MaxCompute allows you to start an Elasticsearch cluster in a MaxCompute cluster by submitting a job. MaxCompute projects do not modify native Elasticsearch code. The Elasticsearch on MaxCompute runs in the same mode as native Elasticsearch clusters.

15.4.2. Limits on Elasticsearch on MaxCompute in VPCs

This topic describes the limits on Elasticsearch on MaxCompute in VPCs.

Elasticsearch on MaxCompute is an enterprise-class retrieval system that is developed by Alibaba Cloud to retrieve large amounts of data. This system also complies with data isolation and security requirements. Therefore, the limits on Elasticsearch on MaxCompute in VPCs are imposed based on the limits of MaxCompute in VPCs.

Elasticsearch on MaxCompute has the following limits in VPCs:

- The classic network, VPCs, and the Internet are isolated from each other. Users can only access the endpoints and virtual IP addresses (VIPs) of their own networks.
- Projects for which VPC IDs or IP address whitelists are not configured are accessible to the three types of networks by using domain names.
- If an Elasticsearch cluster is started in a MaxCompute project, the cluster and project share the same VPC list. The VPC list is a whitelist of VPCs.
- If you start an Elasticsearch cluster, this cluster automatically occupies all resources. If you start more Elasticsearch instances, you must scale up the MaxCompute instance or scale down the Elasticsearch cluster.

Scenario: When you deploy MaxCompute in Apsara Stack, one project is created and one Elasticsearch cluster is started for each project by default. You can start your own Elasticsearch cluster in your project, apply for a domain name and VIP after the cluster is started, and then perform VPC verification in the Elasticsearch frontend.

15.5. Special notes

15.5.1. Find the Elasticsearch service domain name

This topic describes how to find the domain name of the Elasticsearch service during O&M.

1. Log on to the Apsara Infrastructure Management console. In the left-side navigation pane, click Cluster Operations. On the Clusters page, find your MaxCompute project.
2. Click the name of the MaxCompute project. On the page that appears, click the Services tab. On the Services tab, click the name of the desired service.
3. On the Services Details page, find the domain name of the Elasticsearch service.

16. Flink on MaxCompute

This topic provides a demo of Flink on MaxCompute.

In the current MaxCompute version, Flink on MaxCompute is only for trial use. This demo describes only the trial features. For more information about new features, see later versions.

Demo:

1. Modify the configuration file.

Decompress the Flink package, go to the configuration file directory *conf*, and make the following modifications to the *flink-conf.yaml* file:

```
project_name: xxxx
access_id: xxxx
access_key: xxxx
end_point: xxxx
odps.cupid.distributedcache.mincopy: 1
odps.cupid.proxy.domain.name:xxxx
#odps.task.major.version:
cupid_v2
```

Note

- The first four parameters can be obtained from the *odps_config.ini* file under the */home/admin/odps/odps_tools/ctl/conf/* directory.
- *odps.cupid.proxy.domain.name* specifies the domain name of *odps_jobview_server_dns*. The value of this parameter is the domain name without *sparkui*.

2. Start the program.

Save the *flink-java-project-0.1.jar* package to the decompressed directory of Flink and run the following commands:

```
/bin/flink
run -c org.apache.flink.odps.WordCount -m yarn-cluster -yn 1
flink-java-project-0.1.jar --projectName odps_smoke_test --inputTable wc_in
--outputTable wc_out --sleepTime 100
```

3. Log on to Logview and Flink.

Logview



17.Non-structured data access and processing (integrated computing scenarios)

17.1. Overview

This topic describes how to use external tables to access and process unstructured data from internal and external data sources.

MaxCompute SQL cannot directly process external data, such as non-structured data from Object Storage Service (OSS). This type of data must be imported into MaxCompute tables by using relevant tools. This involves complex operations. The MaxCompute team introduced the non-structured data processing framework to the MaxCompute system architecture to simplify the processing of external data.

You can execute a data definition language (DDL) statement to create an external table in MaxCompute and associate the table with external data sources. This table can then act as an interface between MaxCompute and external data sources. External tables can be accessed the same way as standard MaxCompute tables. You can fully use the computing capabilities of MaxCompute SQL to process external data.

MaxCompute allows you to create external tables to process data from the following data sources:

- Internal data sources: OSS, Tablestore, AnalyticDB for MySQL, ApsaraDB RDS, Apsara File Storage for HDFS, and Taobao Distributed Data Layer (TDDL)
- External data sources: open source Hadoop Distributed File System (HDFS), ApsaraDB for MongoDB, and ApsaraDB for HBase

The subsequent topics describe various data sources.

 **Note** MaxCompute V2.0 supports multiple new data types. To use the new data types, you must add `set odps.sql.type.system.odps2=true;` before the SQL statement and commit them to enable the new data types. For ease of reading, sample code in subsequent topics uses new data types by default.

17.2. Internal data sources

17.2.1. OSS data source

17.2.1.1. Overview

This topic describes how to access and process Object Storage Service (OSS) data sources in MaxCompute.

As a core computing component of the Alibaba Cloud big data platform, MaxCompute provides powerful computing capabilities. It can schedule a large number of nodes for parallel computing, and effectively manage the failover and retry mechanisms in a distributed computing environment. MaxCompute SQL implements a variety of logic to process data based on simple semantics. It is widely used within and outside Alibaba Group. It allows interoperability among different data sources and is crucial for building a data ecosystem for Alibaba Cloud.

Some examples are provided to demonstrate how MaxCompute accesses and processes OSS data.

17.2.1.2. Use a built-in extractor to read data from OSS

17.2.1.2.1. Overview

This topic describes how to use a built-in extractor of MaxCompute to read data from Object Storage Service (OSS).

You can use the built-in extractor of MaxCompute. This provides an easy way to read data that is stored in the specified format from OSS. You need only to create an external table as a source table for data queries.

For example, a CSV file is stored in OSS. The endpoint is *oss-cn-shanghai-internal.aliyuncs.com*, the bucket is *oss-odps-test*, and the data file is saved in */demo/SampleData/CSV/AmbulanceData/vehicle.csv*. Some examples are provided to demonstrate how to use the built-in extractor to read data from OSS.

17.2.1.2.2. Create an external table

This topic describes how to create an external table when a built-in extractor is used. It also provides an example for reference.

Execute the following statements to create an external table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
  vehicleId INT,
  recordId INT,
  patientId INT,
  calls INT,
  locationLatitude DOUBLE,
  locationLongitude DOUBLE,
  recordTime STRING,
  direction STRING
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
  'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/';
```

Note

- **STORED BY:** specifies the name of the built-in storage handler. *com.aliyun.odps.CsvStorageHandler* is a built-in storage handler that processes CSV files. It defines how to read data from and write data to CSV files. You can specify this parameter as required. The logic of reading and writing CSV files is implemented by the system.
- **WITH SERDEPROPERTIES:** specifies the properties of the external table that you want to create. *odps.properties.rolearn* specifies the Alibaba Cloud Resource Name (ARN) for the *AliyunODPSDefaultRole* role in Resource Access Management (RAM). You must specify this parameter if you use Security Token Service (STS) to authorize MaxCompute to access OSS.
- **LOCATION:** specifies the OSS directory in which the data files you want to read are saved. By default, the system reads all the data files in the directory.
- An external table contains only related OSS directories. If you delete this table, the data in the directory specified by **LOCATION** is not deleted.

You can execute the following statement to view the structure of the created external table:

```
DESC EXTENDED <table_name>;
```

In the output, you can view basic table information, which is similar to the information returned for a created internal table. In addition to the basic table information, you can view the storage handler and the OSS directory.

17.2.1.2.3. Query an external table

This topic describes how to query an external table when a built-in extractor is used. It also provides an example for reference.

After an external table is created, you can use it the same way that you use a standard table.

In this example, the vehicle.csv file in the `/demo/SampleData/CSV/AmbulanceData/` directory contains the following data:

```
1,1,51,1,46.81006,-92.08174,9/14/2017 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2017 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2017 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2017 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2017 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2017 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2017 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2017 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2017 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2017 0:00,N
```

Execute the following statement to submit a job, which calls a built-in CSV extractor to read data from OSS:

```
SELECT recordId, patientId, direction;
FROM ambulance_data_csv_external;
WHERE patientId > 25;
```

Note

- An external table can be managed only by using MaxCompute SQL, not MaxCompute MapReduce.
- If you want to obtain data over HTTPS at the underlying layer, add `set odps.sql.unstructured.data.oss.use.https=true;` before an SQL statement. Then, commit them for execution.

The following result is returned:

```
+-----+-----+-----+
| recordId | patientId | direction |
+-----+-----+-----+
| 1 | 51 | S |
| 3 | 48 | NE |
| 4 | 30 | W |
| 5 | 47 | S |
| 7 | 53 | N |
| 8 | 63 | SW |
| 10 | 31 | N |
+-----+-----+-----+
```

Note The system provides the following built-in storage handlers: `CsvStorageHandler`, `TsvStorageHandler`, and `TextStorageHandler`.

17.2.1.2.4. MSCK REPAIR TABLE

This topic describes the MSCK REPAIR TABLE statement of MaxCompute and provides an example for reference.

The MSCK REPAIR TABLE statement of MaxCompute can be used to add partitions to external tables. The following example shows the syntax of the MSCK REPAIR TABLE statement:

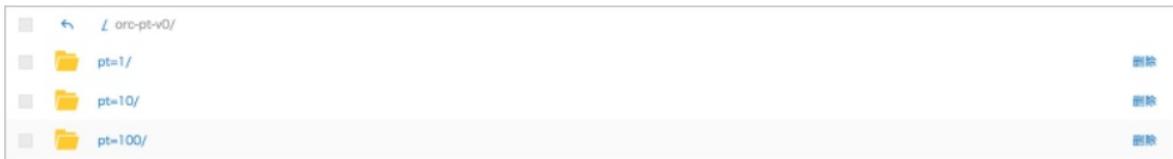
```
MSCK [REPAIR] TABLE external_table_name [ADD PARTITIONS];
```

Syntax description:

- When you import data into Object Storage Service (OSS), make sure that the OSS directory is in the `oss://xxx/table-location/ptname1=ptvalue1/ptname2=ptvalue2/xxx` format.
- You must specify the partition structure when you create an external table.
- After you execute the `MSCK [REPAIR] TABLE external_table_name [ADD PARTITIONS];` statement, MaxCompute SQL automatically parses the OSS directory to identify partitions and adds the partitions to the external table.

Example:

1. Upload data to OSS. The following figure shows the OSS directory.



2. Execute the following statement to create an external table named `orc_pt_v0`. The structure of the `pt` partition is specified in the external table.

```
CREATE EXTERNAL TABLE orc_pt_v0
(
  name STRING
)
PARTITIONED BY (pt bigint)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
  'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
)
LOCATION 'oss://xxx/odps-ext-reg-perf/orc-pt-v0';
```

3. Execute the following statement to add three partitions to the external table `orc_pt_v0`:

```
MSCK REPAIR TABLE orc_pt_v0 ADD PARTITIONS;
-- In this case, the MSCK REPAIR TABLE statement is equivalent to the following three statements
:
ALTER TABLE orc_pt_v0 ADD PARTITION (pt=1);
ALTER TABLE orc_pt_v0 ADD PARTITION (pt=10);
ALTER TABLE orc_pt_v0 ADD PARTITION (pt=100);
```

17.2.1.2.5. Read data from the CSV or TSV files compressed in the GZIP format

This topic describes how to read data from the CSV or TSV files that are compressed in the GZIP format.

MaxCompute can use only a built-in extractor to read data from the CSV and TSV files that are compressed in the GZIP format. The main difference between reading non-compressed and compressed files is the property specified by `SERDEPROPERTIES`.

Note If a data file stored in Object Storage Service (OSS) is an archived object, you must restore the data file first.

The following example shows how to create an external table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
  vehicleId BIGINT,
  recordId BIGINT,
  patientId BIGINT,
  calls BIGINT,
  locationLatitude DOUBLE,
  locationLongitude DOUBLE,
  recordTime STRING,
  direction STRING
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
  'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
  [, 'odps.text.option.gzip.input.enabled'='true']
  [, 'name3'='value3']
)
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/Demo/SampleData/CSV/AmbulanceData/';
```

The following table describes the properties supported by SERDEPROPERTIES.

Property	Valid value	Default value	Description
odps.text.option.gzip.input.enabled	<ul style="list-style-type: none"> True False 	False	Specifies whether to compress the file before data reading.
odps.text.option.gzip.output.enabled	<ul style="list-style-type: none"> True False 	False	Specifies whether to compress the file before data writing.
odps.text.option.header.lines.count	Non-negative integer	0	Specifies the number of rows to skip in the file.
odps.text.option.null.indicator	String	Empty string	Specifies the strings in the file to be parsed as NULL in an SQL statement. For example, if you specify <code>\N</code> to represent NULL, <code>a, \N, b</code> is parsed as <code>a, NULL, b</code> .
odps.text.option.ignore.empty.lines	<ul style="list-style-type: none"> True False 	True	Specifies whether to ignore blank rows.
odps.text.option.encoding	<ul style="list-style-type: none"> UTF-8 UTF-16 US-ASCII GBK 	UTF-8	Specifies the encoding format of the file.
odps.text.option.delimiter	Single character	Comma (,)	Specifies the column delimiter of the file.

Property	Valid value	Default value	Description
odps.text.option.use.quote	<ul style="list-style-type: none"> • True • False 	False	Specifies whether to recognize column delimiters in a CSV file if it uses double quotation marks (") as column delimiters. If fields in the CSV file contain specified symbols for separating multiple values, these fields must be enclosed in double quotation marks ("). The symbols include a carriage return, line feed (CRLF) pair, double quotation mark ("), or comma (,). If a field contains a pair of double quotation marks ("), replace the double quotation marks (") with two pairs of double quotation marks (") for escaping.

 **Note**

If you want to read data from and write data to an external table that is associated with compressed OSS data, configure both `odps.text.option.gzip.input.enabled` and `odps.text.option.gzip.output.enabled` as True when you create the external table.

17.2.1.3. Use a custom extractor to read data from OSS

17.2.1.3.1. Overview

This topic describes how to use a custom extractor to read data from Object Storage Service (OSS).

If OSS data is in a complex format and cannot be processed by the built-in extractor, you must use a custom extractor to read the OSS data.

For example, a text file is stored in OSS. The columns of data records in the file are separated by vertical bars (|). The data file `vehicle.csv` is saved in the `/demo/SampleData/CustomTxt/AmbulanceData/` directory. Some examples are provided to demonstrate how to use a custom extractor to read data from OSS.

17.2.1.3.2. Define a storage handler

This topic describes how to define a storage handler when a custom extractor is used. It also provides an example for reference.

You can customize the logic to parse data. `StorageHandler` is the unified entrance of your custom logic. You can use `StorageHandler` to specify the types of custom extractors and `Outputer`. `StorageHandler` provides only a simple definition. For example, you can customize a `SpeicalTextStorageHandler`:

```
package com.aliyun.odps.udf.example.text;
public class SpeicalTextStorageHandler extends OdpsStorageHandler {
    @Override
    public Class<? extends Extractor> getExtractorClass() {
        return TextExtractor.class;
    }
    @Override
    public Class<? extends Outputter> getOutputterClass() {
        return TextOutputter.class;
    }
}
```

 **Note** TextStorageHandler that is built-in to MaxCompute can process the text data in which columns are separated by vertical bars (|). This example demonstrates how to customize a storage handler by using an SDK to process specially formatted data, especially in scenarios in which you use an extractor.

17.2.1.3.3. Define an extractor

This topic describes how to define a custom extractor when you want to use this extractor. It also provides an example for reference.

In the following example, TextExtractor is used to extract records from a text file, where the delimiter is imported as a parameter. TextExtractor can be used for all text files of the similar format.

```
/**
 * Text extractor that extract schematized records from formatted plain-
 text(csv, tsv etc.)
 */
public class TextExtractor extends Extractor {
private InputStreamSet inputs;
private String columnDelimiter;
private DataAttributes attributes;
private BufferedReader currentReader;
private boolean firstRead = true;
public TextExtractor() {
// default to ",", this can be overwritten if a specific delimiter is
provided (via DataAttributes)
this.columnDelimiter = ",";
}
// no particular usage for execution context in this example
@Override
public void setup(ExecutionContext ctx, InputStreamSet inputs,
DataAttributes attributes) {
this.inputs = inputs;
-- inputs specifies an InputStreamSet. An InputStream is returned each time the next() method is cal
led. This InputStream can read all data from an OSS file.
this.attributes = attributes;
// check if "delimiter" attribute is supplied via SQL query
String columnDelimiter = this.attributes.getValueByKey("delimiter");
-- The delimiter can be used as a parameter in DDL statements.
if ( columnDelimiter != null)
{
this.columnDelimiter = columnDelimiter;
}
// note: more properties can be inited from attributes if needed
}
@Override
public Record extract() throws IOException {
String line = readNextLine();
if (line == null) {
return null;
-- If null is returned, all records in the table have been read.
}
return textLineToRecord(line);
-- textLineToRecord splits a row into multiple columns by using the delimiter. For more information
about the implementation process, see Complete implementation of TextExtractor.
-- extractor() returns a record that is extracted from OSS data.
}
@Override
public void close(){
// no-op
}
}
```

17.2.1.3.4. Compile and package code

This topic describes how to compile and package code when you use a custom extractor.

You can compile and package Java code, and run the following command to upload the package to MaxCompute. The procedure is the same as that for a common Java user-defined function (UDF).

```
add jar odps-udf-example.jar;
```

17.2.1.3.5. Create an external table

This topic describes how to create an external table when a custom extractor is used. It also provides an example for reference.

After you upload a JAR package, you must run the following command to create an external table. This command is similar to that you run when a built-in extractor is used. The difference is that a custom storage handler is used in this command.

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_txt_external
(
  vehicleId int,
  recordId int,
  patientId int,
  calls int,
  locationLatitude double,
  locationLongitude double,
  recordTime string,
  direction string
)
STORED BY 'com.aliyun.odps.udf.example.text.SpecialTextStorageHandler'
-- STORED BY specifies the class name of a custom storage handler.
WITH SERDEPROPERTIES('delimiter'=',')
-- SERDEPROPERTIES can be used to specify parameters. These parameters are transferred to an extractor
or by using DataAttributes.
LOCATION 'oss://<your-id*>:<your-secret-key*>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/SampleData/CustomTxt/AmbulanceData/'
USING 'odps-udf-example.jar';
-- Specify the JAR package in which the class definition is located.
```

17.2.1.3.6. Query an external table

This topic describes how to query an external table when a custom extractor is used. It also provides an example for reference.

In this example, the vehicle.csv file in the `/demo/SampleData/CustomTxt/AmbulanceData/` directory contains the following data:

```
1|1|51|1|46.81006|-92.08174|9/14/2017 0:00|S
1|2|13|1|46.81006|-92.08174|9/14/2017 0:00|NE
1|3|48|1|46.81006|-92.08174|9/14/2017 0:00|NE
1|4|30|1|46.81006|-92.08174|9/14/2017 0:00|W
1|5|47|1|46.81006|-92.08174|9/14/2017 0:00|S
1|6|9|1|46.81006|-92.08174|9/14/2017 0:00|S
1|7|53|1|46.81006|-92.08174|9/14/2017 0:00|N
1|8|63|1|46.81006|-92.08174|9/14/2017 0:00|SW
1|9|4|1|46.81006|-92.08174|9/14/2017 0:00|NE
1|10|31|1|46.81006|-92.08174|9/14/2017 0:00|N
```

Execute the following statement to submit a job, which calls a custom extractor to read data from OSS:

```
SELECT recordId, patientId, direction;
FROM ambulance_data_txt_external;
WHERE patientId > 25;
```

The following result is returned:

```
+-----+-----+-----+
| recordId | patientId | direction |
+-----+-----+-----+
| 1 | 51 | S |
| 3 | 48 | NE |
| 4 | 30 | W |
| 5 | 47 | S |
| 7 | 53 | N |
| 8 | 63 | SW |
| 10 | 31 | N |
+-----+-----+-----+
```

17.2.1.4. Use a custom extractor to read external unstructured data

This topic describes how to use a custom extractor to read external unstructured data. It also provides an example for reference.

The preceding topics describe how to use built-in and custom extractors to process text files such as CSV files that are stored in OSS. This topic describes how to use a custom extractor to process non-text files that are stored in OSS.

Audio files in the WAV format are used in this example.

1. Customize the `SpeechSentenceSnrExtractor` main logic. Call the `setup` method to read parameters, initialize the parameters, and import the audio processing model. You can use the `resource` function to import the model.

```
public SpeechSentenceSnrExtractor(){
    this.utteranceLabels = new HashMap<String, UtteranceLabel>();
}
@Override
public void setup(ExecutionContext ctx, InputStreamSet inputs,
    DataAttributes attributes){
    this.inputs = inputs;
    this.attributes = attributes;
    this.mlfFileName = this.attributes.getValueByKey(MLF_FILE_ATTRIBUTE_KEY);
    String sampleRateInKHzStr =
    this.attributes.getValueByKey(SPEECH_SAMPLE_RATE_KEY);
    this.sampleRateInKHz = Double.parseDouble(sampleRateInKHzStr);
    try {
        // read the speech model file from resource and load the model into
        memory
        BufferedInputStream inputStream =
        ctx.readResourceFileAsStream(mlfFileName);
        loadMlfLabelsFromResource(inputStream);
        inputStream.close();
    } catch (IOException e) {
        throw new RuntimeException("reading model from mlf failed with exception
        " + e.getMessage());
    }
}
```

```

    }
    @Override
    public Record extract() throws IOException {
        SourceInputStream inputStream = inputs.next();
        if (inputStream == null){
            return null;
        }
        // process one wav file to extract one output record [snr, id]
        String fileName = inputStream.getFileName();
        fileName = fileName.substring(fileName.lastIndexOf('/') + 1);
        logger.info("Processing wav file " + fileName);
        // infer id from speech file name
        String id = fileName.substring(0, fileName.lastIndexOf('.'));
        // read speech file into memory buffer
        long fileSize = inputStream.getFileSize();
        byte[] buffer = new byte[(int)fileSize];
        int readSize = inputStream.readToEnd(buffer);
        inputStream.close();
        // compute the avg sentence snr from speech file
        double snr = computeSnr(id, buffer, readSize);
        // construct output record [snr, id]
        Column[] outputColumns = this.attributes.getRecordColumns();
        ArrayRecord record = new ArrayRecord(outputColumns);
        record.setDouble(0, snr);
        record.setString(1, id);
        return record;
    }
    private void loadMlflabelsFromResource(BufferedInputStream fileInputStream)
        throws IOException {
        // loading MLF label from resource, skipped here
    }
    // compute the snr of the speech sentence, assuming the input buffer
    // contains the entire content of a wav file
    private double computeSnr(String id, byte[] buffer, int validBufferLen){
        // computing the snr value for the wav file (supplied as byte buffer
        // array), skipped here
    }
}

```

Note The `extract()` method implements the reading and processing logic of audio files. It calculates the signal-to-noise ratio (SNR) of the read data based on the audio processing model and writes the results to a record in the `[snr, id]` format.

2. Run the following commands to create an external table:

```

CREATE EXTERNAL TABLE IF NOT EXISTS speech_sentence_snr_external
(
    sentence_snr double,
    id string
)
STORED BY 'com.aliyun.odps.udf.example.speech.SpeechStorageHandler'
WITH SERDEPROPERTIES (
    'mlfFileName'='sm_random_5_utterance.text.label' ,
    'speechSampleRateInKHz' = '16'
)
LOCATION 'oss://<your-id>:<your-secret-key>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/dev/SpeechSentenceTest/'
USING 'odps-udf-example.jar,sm_random_5_utterance.text.label';

```

3. Run the following commands to read data from OSS:

```
SELECT sentence_snr, id
FROM speech_sentence_snr_external
WHERE sentence_snr > 10.0;
```

4. View the processing results:

```
-----
| sentence_snr | id |
-----
| 34.4703 | J310209090013_H02_K03_042 |
-----
| 31.3905 | tsh148_seg_2_3013_3_6_48_80bd359827e24dd7_0 |
-----
| 35.4774 | tsh148_seg_3013_1_31_11_9d7c87aef9f3e559_0 |
-----
| 16.0462 | tsh148_seg_3013_2_29_49_f4cb0990a6b4060c_0 |
-----
| 14.5568 | tsh_148_3013_5_13_47_3d5008d792408f81_0 |
-----
```

 **Note** A custom extractor allows you to execute SQL statements to process multiple audio files in OSS in a distributed manner. Similarly, you can use the large-scale computation capabilities of MaxCompute to process unstructured data, such as images and videos.

17.2.1.5. Data partitions

17.2.1.5.1. Overview

This topic describes the data partitioning feature of external tables.

LOCATION is used to specify an OSS directory, which is associated with an external table. MaxCompute reads all data in the OSS directory, including all files in the subdirectories. Due to a large amount of data accumulated in the directory, a full-text scan is performed. This operation results in extra I/O operations and prolongs the time required to process data. Two solutions are provided to address this issue:

- Reduce the amount of data: Plan data storage directories properly. Create multiple external tables for data from different parts, with LOCATION of each external table pointing to a subset of data.
- Partition data: Similar to internal tables, external tables support data partitioning. You can create partitions to manage the data.

Some examples are provided to demonstrate how to use the data partitioning feature of external tables.

17.2.1.5.2. Standard organization method and directory structure of partition data in OSS

This topic describes the standard organization method and directory structure of partition data in Object Storage Service (OSS). It also provides an example for reference.

Unlike the data in the internal tables of MaxCompute, the data stored in external storage, such as OSS, cannot be managed in MaxCompute. If you want to use the partitioned table feature of MaxCompute, make sure that the directories of data files in OSS are in the following format:

```
partitionKey1=value1\partitionKey2=value2\...
```

Example:

1. Your daily log files are stored in OSS, and you want to access some of the data from MaxCompute on a daily basis. If the log files are in the CVS format or a similar custom format, you can execute the following statement to create a partitioned external table:

```
CREATE EXTERNAL TABLE log_table_external (  
  click STRING,  
  ip STRING,  
  url STRING,  
)  
PARTITIONED BY (  
  year STRING,  
  month STRING,  
  day STRING  
)  
STORED BY 'com.aliyun.odps.CsvStorageHandler'  
LOCATION 'oss://<ak_id>:<ak_key>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/log_data/';
```

Note In the preceding example, the PARTITIONED BY clause is used to specify a partitioned external table. The partition keys are year, month, and day.

2. To make the partitions take effect, you must specify the OSS storage directory in the format shown in the preceding example. The following example shows a valid directory layout:

```
osscmd ls oss://oss-odps-test/log_data/  
2017-09-14 08:03:35 128MB Standard oss://oss-odps-  
test/log_data/year=2017/month=06/day=01/logfile  
2017-09-14 08:04:12 127MB Standard oss://oss-odps-  
test/log_data/year=2017/month=06/day=01/logfile.1  
2017-09-14 08:05:02 118MB Standard oss://oss-odps-  
test/log_data/year=2017/month=06/day=02/logfile  
2017-09-14 08:06:45 123MB Standard oss://oss-odps-  
test/log_data/year=2017/month=07/day=10/logfile  
2017-09-14 08:07:11 115MB Standard oss://oss-odps-  
test/log_data/year=2017/month=08/day=08/logfile  
...
```

Note If you prepare offline data, use a tool to upload the offline data to OSS. In this case, you can specify the data directory format. For the partitioned table feature of the external table to work properly, we recommend that you use the directory format in the preceding example for the uploaded data.

3. You can execute the ALTER TABLE ADD PARTITION statement to import the partition information to MaxCompute. The following example shows sample statements:

```
ALTER TABLE log_table_external ADD PARTITION (year = '2017', month = '06', day = '01')  
ALTER TABLE log_table_external ADD PARTITION (year = '2017', month = '06', day = '02')  
ALTER TABLE log_table_external ADD PARTITION (year = '2017', month = '07', day = '10')  
ALTER TABLE log_table_external ADD PARTITION (year = '2017', month = '08', day = '08')  
...
```

4. If the data is ready and the partition information is imported to MaxCompute, you can execute SQL statements to manage the partitions in the external table in OSS.
 - o Execute the following statement to count the number of unique IP addresses in the log generated on June 1, 2017:

```
SELECT count(distinct(ip)) FROM log_table_external;
WHERE year = '2017' AND month = '06' AND day = '01';
```

Note In this example, MaxCompute accesses only logfile and logfile.1 in the *log_data/year=2016/month=06/day=01* sub-folder but not all files in the *log_data* folder for the external table *log_table_external*. This prevents unnecessary I/O operations.

- Similarly, you can execute the following statement to analyze data from the second half of year 2017:

```
SELECT count(distinct(ip)) FROM log_table_external;
WHERE year = '2017' AND month > '06';
```

Note In this example, only the logs for the second half of year 2017 stored in OSS are accessed.

17.2.1.5.3. Custom directories of partition data in OSS

This topic describes the custom directories in which partition data is saved in Object Storage Service (OSS). It also provides an example for reference.

If you have historical data stored in OSS but the data is not saved in the *partitionKey1=value1\partitionKey2=value2\...* format, you can still access the data by using the partitioning feature of MaxCompute. MaxCompute provides a way to import partitions from custom directories.

Example:

1. The directories in which partition data is saved contain only partition values but do not contain partition keys. The following example shows the layout of the data directories:

```
osscmd ls oss://oss-odps-test/log_data_customized/
2017-09-14 08:03:35 128MB Standard oss://oss-odps-
test/log_data_customized/2017/06/01/logfile
2017-09-14 08:04:12 127MB Standard oss://oss-odps-
test/log_data_customized/2017/06/01/logfile.1
2017-09-14 08:05:02 118MB Standard oss://oss-odps-
test/log_data_customized/2017/06/02/logfile
2017-09-14 08:06:45 123MB Standard oss://oss-odps-
test/log_data_customized/2017/07/10/logfile
2017-09-14 08:07:11 115MB Standard oss://oss-odps-
test/log_data_customized/2017/08/08/logfile
...
```

2. You can run the following command to bind subdirectories to different partitions:

```
ALTER TABLE log_table_external ADD PARTITION (year = '2017', month = '06', day = '01')
LOCATION 'oss://<ak_id>:<ak_key>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/log_data_cu
stomized/2017/06/01/';
```

Note The LOCATION parameter specifies an OSS directory. This directory maps to the partition that is added to the external table by using ADD PARTITION. This way, MaxCompute can access data in the OSS directory even if the directory is not in the *partitionKey1=value1\partitionKey2=value2\...* format. In addition, you can access the partition data in the subdirectory.

17.2.1.5.4. Access fully-customized non-partitioned data subsets

This topic describes how to access fully-customized non-partitioned data subsets.

In some special cases, you may need to access an arbitrary subset of a file in an OSS directory, and the format of the directory where the files in this file subset are saved is not deterministic. The unstructured data processing framework of MaxCompute is able to handle this situation, but is not discussed in this topic.

For more information about advanced operations, contact the MaxCompute technical team.

17.2.1.6. Output OSS data

17.2.1.6.1. Create an external table

This topic describes how to create an external table when you write data to OSS. It also provides an example for reference.

To write data to OSS, you must execute the CREATE EXTERNAL TABLE statement to create an external table first. The process is the same as that of reading data from OSS. After the external table is created, you can execute MaxCompute SQL statements such as INSERT INTO or OVERWRITE. The following example demonstrates how to create an external table by using the built-in storage handler TsvStorageHandler of MaxCompute.

```
DROP TABLE IF EXISTS tpch_lineitem_tsv_external;
CREATE EXTERNAL TABLE IF NOT EXISTS tpch_lineitem_tsv_external
(
  orderkey BIGINT,
  suppkey BIGINT,
  discount DOUBLE,
  tax DOUBLE,
  shipdate STRING,
  linestatus STRING,
  shipmode STRING,
  comment STRING
)
STORED BY 'com.aliyun.odps.TsvStorageHandler'
LOCATION 'oss://<AK_id>:<AK_secret>@oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/tsv_output_folder
/';
```

Note

The preceding DDL statement creates an external table named tpch_lineitem_tsv_external, and associates two external data dimensions with this external table.

- Data storage medium: LOCATION associates an OSS address with the external table. This address is used to read data from or write data to the external table.
- Data storage format: A storage handler is used to define the data access mode. In this example, the built-in storage handler com.aliyun.odps.TsvStorageHandler of MaxCompute is used to read or write data from or to TSV files. You can also use the MaxCompute SDK to define storage handlers.

17.2.1.6.2. Write data to a TSV text file by using an INSERT statement on an external table

This topic describes how to write data to a TSV text file by using an INSERT statement on an external table. It also provides an example for reference.

After you associate an OSS object with an external table, you can execute a standard INSERT OVERWRITE or INSERT INTO statement on the external table to write data to the OSS file. The data source can be either the data stored in MaxCompute internal tables or the external data that is imported into MaxCompute by using an external table.

Note

- MaxCompute internal table: You can execute an INSERT statement on an external table to write data from a MaxCompute internal table to an external storage medium.
- External data that is imported into MaxCompute by using an external table: You can import external data to MaxCompute by using an external table, use the data for computations, and then export the results to an external storage directory or external data store. For example, you can import Tablestore data into MaxCompute and then export the data to OSS.

In this example, you have a MaxCompute internal table named `tpch_lineitem` and want to export some of the data in the table to OSS in the TSV format. After you create an external table, you can execute the INSERT OVERWRITE statement to export the data:

```
INSERT OVERWRITE TABLE tpch_lineitem_tsv_external;  
SELECT l_orderkey, l_suppkey, l_discount, l_tax, l_shipdate, l_linestatus,  
l_shipmode, l_comment  
FROM tpch_lineitem  
WHERE l_discount = 0.07 and l_tax = 0.01;
```

In the preceding example, eight columns are selected from the rows that meet the conditions `l_discount = 0.07` and `l_tax = 0.01` in the internal table `tpch_lineitem` and then written to OSS in the TSV format. The selected columns in the internal table correspond to the schema of the external table `tpch_lineitem_tsv_external`. After this operation is complete, you can view the related TSV data file in OSS.

Notice

Data exported from MaxCompute to OSS is stored in a special file structure.

- If you use MaxCompute to execute the INSERT INTO or INSERT OVERWRITE statement on an external table and write data to an OSS directory, all data is saved in a `.odps` folder in the directory specified by `LOCATION`.
- The `.meta` file in the `.odps` folder is an extra macro data file written by MaxCompute. This file records valid data in the folder. If the INSERT operation succeeds, all data in the folder is valid. You are only required to parse the macro data if a job fails.
- If a job fails or is terminated, execute the INSERT OVERWRITE statement again until it is complete. This prevents the `.meta` file from being parsed.
- If you want to parse the `.meta` file, contact Alibaba Cloud technical team.

The number of files that are generated during the internal processing of TSV and CSV files of MaxCompute is equal to the number of concurrent SQL stages. You can use the flexible semantics and configurations of MaxCompute to limit the number of generated files. In the preceding example, if you want to force the generation of a TSV file, you can append `DISTRIBUTE BY l_discount` to the INSERT OVERWRITE statement. Then, a reduce stage with only one reducer is added at the end so that only one TSV file is generated.

17.2.1.6.3. Write data to an unstructured file by using an INSERT statement on an external table

This topic describes how to write data to an unstructured file by using an INSERT statement on an external table.

MaxCompute provides the Outputter interface for data output. You can call this interface to write user data to a custom unstructured data file by using OutputStream. Further details are not provided in this topic.

If you have related requirements, contact the MaxCompute technical team.

17.2.1.6.4. Migrate data between different storage media by using MaxCompute

This topic describes how to migrate data between different storage media by using MaxCompute. It also provides an example for reference.

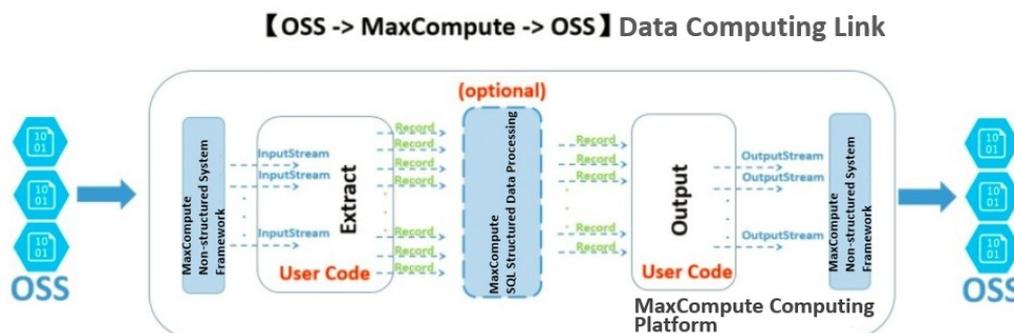
External tables act as an interface between MaxCompute and external storage media. External tables can be used to read data from or write data to various external storage media. Based on the feature of external tables, various links can be established to compute and store data. For example:

- MaxCompute reads data from an Object Storage Service (OSS) bucket that is associated with External table A, performs complex computations, and then generates results to the directory of the OSS bucket that is associated with External table B.
- MaxCompute reads data from Tablestore that is associated with External table A, performs complex computations, and then generates results to the directory of the OSS bucket that is associated with External table B.

Note In the preceding examples, the data source for the SELECT statement is an external table, rather than a MaxCompute internal table.

The following figure shows the flowchart for data migration.

MaxCompute is used as a central computing platform. It reads data from an OSS bucket and then writes the data to another OSS bucket (in a different location or within a different OSS account).



Based on the data flow and processing logic shown in the preceding figure, the unstructured data processing framework can be considered a coupled data ingress and egress at both ends of MaxCompute.

1. The external data from an OSS bucket is converted based on the unstructured framework, and provided to the InputStream class of Java. The extract logic is only used to read, parse, transform, and compute the data from the InputStream class, and return the data in the Record format used by MaxCompute.
2. Part of the returned records are used in the SQL logical operations on MaxCompute. These operations utilize

the powerful SQL computation engine that is built in MaxCompute, and may generate new records.

3. The records obtained after computations are transferred to your custom output logic for further computations. Finally, the required information is extracted from the records, transferred by using OutputStream, and written to OSS.

 **Note** You can perform the preceding steps based on your business requirements.

17.2.1.7. STS mode authorization for OSS

This topic describes the Security Token Service (STS) mode authorization for Object Storage Service (OSS), and provides an example for reference.

When you create an external table, the Location-based OSS access account allows you to enter AccessKey ID and AccessKey secret in plaintext mode. However, the account information may be exposed. To prevent account information from being exposed, MaxCompute provides a more secure way to access OSS.

MaxCompute integrates RAM and STS of Alibaba Cloud to enhance account security. You can use one of the following methods to grant permissions:

- If the owners of MaxCompute and OSS use the same Alibaba Cloud account, you can authorize access to OSS with one click in the RAM console.
- Custom authorization is supported.
 - i. You can log on to the RAM console and authorize access to OSS.

Create a role named AliyunODPSDefaultRole or AliyunODPSRoleForOtherUser and compile the following policy content:

```
-- If the owners of MaxCompute and OSS use the same Alibaba Cloud account:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}

-- If the owners of MaxCompute and OSS use different Alibaba Cloud accounts:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ID of the Alibaba Cloud account that owns the MaxCompute project@odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

- ii. Grant the AliyunODPSRolePolicy permission, which is required for the role to access OSS.

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "oss:ListBuckets",
        "oss:GetObject",
        "oss:ListObjects",
        "oss:PutObject",
        "oss:DeleteObject",
        "oss:AbortMultipartUpload",
        "oss:ListParts"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
-- You can also add other permissions as required.
```

- iii. Grant the AliyunODPSRolePolicy permission to the role.

 **Note** After authorization is complete, view the role details to obtain the RAM information of this role. You must specify the RAM information when you create an OSS external table later.

17.2.2. Tablestore data source

17.2.2.1. Overview

This topic describes how to access and process data of Tablestore data sources in MaxCompute.

As the core computing component of the Alibaba Cloud big data platform, MaxCompute meets most distributed computing requirements within and outside Alibaba Group. MaxCompute SQL provides powerful support for the quick processing and storage of exabytes of offline data. As big data businesses continue to grow, many new data usage scenarios emerge. To adapt to the new scenarios, the MaxCompute computing framework is constantly evolving. Its powerful computation capabilities, originally designed to process internal data in special formats, have expanded to process external data sources in various formats. This topic describes how to import data from Tablestore to MaxCompute, which implements seamless interoperability between data sources.

In online service scenarios, NoSQL KV storage (such as Bigtable or HBase) features flexible schema, easy scalability, and high real-time performance, compared with traditional databases. Alibaba Cloud Tablestore is a large-scale NoSQL data storage service based on the Apsara distributed operating system. It stores and allows real-time access to large volumes of key-value pairs. Tablestore is widely used by all business units in Alibaba Group and the Alibaba Cloud ecosystem. In particular, Tablestore features, such as row-level real-time update and override writing, supplement the append-only operations of MaxCompute tables. As a storage-oriented service, Tablestore does not provide sufficient computing capabilities to concurrently process large volumes of data. In this case, MaxCompute allows you to create external tables to access Tablestore data sources.

Examples are provided to demonstrate how MaxCompute accesses and processes Tablestore data.

 **Note** This topic assumes that you have a basic knowledge of Tablestore operations. If you are not familiar with Tablestore or are new to the concept of key-value tables, you can first learn some basic Tablestore concepts, such as primary keys, partition keys, and attribute columns.

17.2.2.2. Use MaxCompute to read and calculate data in

Tablestore

17.2.2.2.1. Create an external table

This topic describes how to create an external table for Tablestore data sources. It also provides an example for reference.

MaxCompute accesses Tablestore data by using external tables. After you execute the `CREATE EXTERNAL TABLE` statement to introduce the description of Tablestore table data to the metadata of MaxCompute, you can process Tablestore data in the same way as you process data in a standard table.

Execute the following statements to create an external table:

```
DROP TABLE IF EXISTS ots_table_external;
CREATE EXTERNAL TABLE IF NOT EXISTS ots_table_external
(
  odps_orderkey bigint,
  odps_orderdate string,
  odps_custkey bigint,
  odps_orderstatus string,
  odps_totalprice double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
-- com.aliyun.odps.TableStoreStorageHandler is a MaxCompute built-in storage handler that processes
Tablestore data. It defines the interaction between MaxCompute and Tablestore. The related logic is
implemented by MaxCompute.
WITH SERDEPROPERTIES (
  -- SERDEPROPERTIES is considered an interface that provides parameter options. Two options must be s
pecified for com.aliyun.odps.TableStoreStorageHandler: tablestore.columns.mapping and tablestore.tab
le.name.
'tablestore.columns.mapping'=':o_orderkey, :o_orderdate, o_custkey,
o_orderstatus,o_totalprice',
-- tablestore.columns.mapping: This option is required. It describes the columns of Tablestore table
s that are accessed by MaxCompute. The columns include primary key columns and attribute columns. Co
lumn names whose names start with a colon (:) are primary key columns in Tablestore tables. In this examp
le, :o_orderkey and :o_orderdate are primary key columns. Other columns are attribute columns. A Tab
lestore table can have one to four primary keys of the BIGINT or STRING type. The first primary key
is the partition key. If you specify a table mapping, you must provide all primary key columns in th
e specified Tablestore table. You can specify only the attribute columns that are accessed by MaxCom
pute.
'tablestore.table.name'='ots_tpch_orders'
-- tablestore.table.name: This option is required. It describes the names of Tablestore tables that
are accessed by MaxCompute. If you specify an invalid (nonexistent) Tablestore table name, an error
is returned and MaxCompute does not create a Tablestore table with this name.
)
LOCATION 'tablestore://<your AK id*>:<your AK secret key*>@odps-ots-dev.cn-
hangzhou.ots.aliyuncs.com';
-- The LOCATION clause specifies the Tablestore information, including the instance name and endpoint.
```

 **Note** The preceding example maps a Tablestore table to an external table of MaxCompute. After the mapping is specified, you can perform subsequent operations on Tablestore data by using the external table.

17.2.2.2.2. Use an external table to access Tablestore data

This topic describes how to use an external table to access TableStore data and perform computations.

After you create an external table, Tablestore data is introduced into the MaxCompute ecosystem. Then, you can access Tablestore data by using the MaxCompute SQL syntax.

Example:

```
SELECT odps_orderkey, odps_orderdate, SUM(odps_totalprice) AS totalprice
FROM ots_table_external
WHERE odps_orderkey > 5000 AND odps_orderdate >20170725 AND odps_orderdate <20170910
GROUP BY odps_orderkey, odps_orderdate
HAVING totalprice> 2000;
```

Note In this example, the MaxCompute SQL syntax that you are familiar with is directly used, and all the other implementations of accessing Tablestore data are performed by MaxCompute.

If you want to compute one piece of data multiple times, you can import the data from Tablestore into an internal table of MaxCompute. This way, you do not need to read the data from Tablestore each time you compute the data.

Example:

```
CREATE TABLE internal_orders AS
SELECT odps_orderkey, odps_orderdate, odps_custkey, odps_totalprice
FROM ots_table_external
WHERE odps_orderkey > 5000 ;
```

Note In this example, internal_orders is a MaxCompute table that you are familiar with. This table has all features of MaxCompute internal tables, such as efficient column compression for data storage and complete metadata. This table is stored in MaxCompute. Therefore, it can be accessed faster than an external table in Tablestore. This feature is particularly suitable for hotspot data that is used for multiple computations.

17.2.2.3. Write data from MaxCompute to Tablestore

This topic describes how to use external tables to write data from MaxCompute to Tablestore.

Data interaction between MaxCompute and Tablestore includes importing data from Tablestore to MaxCompute for batch processing and exporting the data processing results from MaxCompute to Tablestore. Tablestore features, such as real-time update and single-row overwriting, allow you to quickly upload offline computing results to online applications. The `INSERT OVERWRITE` statement is used to export data processing results from MaxCompute to Tablestore.

Note MaxCompute does not proactively create external tables in Tablestore. Before you export data to a table in Tablestore, make sure that the table exists in Tablestore. If the table does not exist, an error is returned.

For example, you have created the external table `ots_table_external` in MaxCompute to access data of the `ots_tpch_orders` table in Tablestore. A data record named `internal_orders` is stored in MaxCompute. To process the data record `internal_orders` and write the processing results to Tablestore, execute the `INSERT OVERWRITE TABLE` statement. Sample statement:

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custkey,
'SHIPPED'), CEIL(odps_totalprice)
FROM internal_orders;
```

Note

- Tablestore is a NoSQL service that stores data in the format of key-value pairs. Data outputs from MaxCompute affect only the rows that contain the primary keys of the Tablestore table. In addition, only the attribute columns specified when you create the table are updated. The columns that are not included in the external table are not modified.
- If you execute the `INSERT OVERWRITE` statement on the external table, MaxCompute inserts 200 data records into the table at a time by default. You can adjust a batch size to limit the total batch size to 4 MB.

17.2.3. AnalyticDB data source

17.2.3.1. Overview

This topic describes how to access and process data of AnalyticDB for MySQL data sources in MaxCompute.

AnalyticDB for MySQL updates or processes data. If both the data processed by AnalyticDB for MySQL and the data in MaxCompute are used for data computations, the data from AnalyticDB for MySQL must be synchronized with the data from MaxCompute. Therefore, you must create an external table to access the data of AnalyticDB for MySQL data sources.

The following example shows how MaxCompute accesses and processes the data of AnalyticDB for MySQL data sources.

17.2.3.2. Write data to AnalyticDB

17.2.3.2.1. Create an external table

This topic describes how to create external tables for AnalyticDB for MySQL data sources and provides an example for reference.

Run the following command to create an external table:

```
set odps.sql.hive.compatible=true;
drop table if exists ads_table_external;
CREATE EXTERNAL TABLE if not exists ads_table_external
(
  c_int int ,
  c_tinyint tinyint ,
  c_boolean boolean ,
  c_smallint smallint ,
  c_bigint bigint ,
  c_double double ,
  c_float float ,
  --c_time datetime ,
  c_date date ,
  c_timestamp datetime ,
  c_string string
)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
location 'jdbc:mysql://host:port/databasename?useSSL=false&user=${user}&password=${password}&table=${tablename}'
```

17.2.3.2.2. Write and query data

This topic describes how to use external tables to write and query AnalyticDB for MySQL data.

After an external table is created, you can use it the same way that you use a standard table. You can use the INSERT OVERWRITE, INSERT INTO, and SELECT statements to write data to the external table and check whether the write operation succeeds. For more information about the usage of these statements, see [Update the data of a table](#) and [SELECT](#).

17.2.3.3. Read data from AnalyticDB for MySQL

This topic describes how to read data from AnalyticDB for MySQL and provides an example for reference.

If you have compiled the ads_read_external script, execute the following statements to read data from AnalyticDB for MySQL:

```
set odps.sql.hive.compatible=true;
drop table if exists ads_read_external;
CREATE EXTERNAL TABLE if not exists ads_read_external
(
  c_int int ,
  c_tinyint tinyint ,
  c_boolean boolean ,
  c_smallint smallint ,
  c_bigint bigint ,
  c_double double ,
  c_float float ,
  --c_time datetime ,
  c_date date ,
  c_timestamp datetime ,
  c_string string
)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
location 'jdbc:mysql://host:port/databasename?useSSL=false&user=${user}&password=${password}&table=${tablename}'
-- Create an external table.
select * from ads_read_external;
-- Query and read data.
```

17.2.4. RDS data source

17.2.4.1. Overview

This topic describes how to access and process data of ApsaraDB RDS data sources in MaxCompute.

ApsaraDB RDS updates or processes data. If both the data processed by ApsaraDB RDS and the data in MaxCompute are used for computations, the data from ApsaraDB RDS must be synchronized with the data from MaxCompute. Therefore, you must create an external table to access the ApsaraDB RDS data source.

Some examples are provided to demonstrate how MaxCompute accesses and processes data of ApsaraDB RDS data sources.

Note When you create an external table, you are not required to create the related table in ApsaraDB RDS. However, when you perform the SELECT or INSERT operation on external tables, you must first create the related tables in ApsaraDB RDS.

17.2.4.2. Write data to RDS

17.2.4.2.1. Create an external table

This topic describes how to create an external table for ApsaraDB RDS data sources. It also provides an example for reference.

Execute the following statements to create an external table:

Note ApsaraDB RDS for MySQL versions earlier than V8.0 are supported. If the ApsaraDB RDS for MySQL version is 8.0 or later, the error `FAILED: Generating job conf failed, gen jobconf failed` is returned. In this case, the external table cannot be created.

```
set odps.sql.hive.compatible=true;
drop table if exists rds_table_external;
CREATE EXTERNAL TABLE if not exists rds_table_external
(
  id bigint,
  name string,
  age tinyint
)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
location 'jdbc:mysql://host:port/databasename?useSSL=false&user=${user}&password=${password}&table=${tablename}'
```

17.2.4.2.2. Write and query data

This topic describes how to use external tables to write and query ApsaraDB RDS data.

After an external table is created, you can use it the same way you use a common data table. You can execute the `INSERT OVERWRITE`, `INSERT INTO`, and `SELECT` statements to write data and check whether the write operation succeeds. For more information about the usage of these statements, see [Update the data of a table](#) and [SELECT](#).

17.2.4.3. Read data from ApsaraDB RDS

This topic describes how to read data from ApsaraDB RDS and provides an example for reference.

If you have compiled the `rds_read_external` script, run the following commands to read data from ApsaraDB RDS:

```
set odps.sql.hive.compatible=true;
drop table if exists rds_read_external;
CREATE EXTERNAL TABLE if not exists rds_read_external
(
  id int,
  name string,
  age int
)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
location 'jdbc:mysql://host:port/databasename?useSSL=false&user=${user}&password=${password}&table=${tablename}'
-- Create an external table.
select * from rds_read_external;
-- Query and read data from the external table.
```

17.2.5. HDFS data source (Alibaba Cloud)

17.2.5.1. Overview

This topic describes how to access and process Apsara File Storage for HDFS data sources in MaxCompute.

Apsara File Storage for HDFS is a distributed file system designed for Alibaba Cloud computing resources such as Elastic Compute Service (ECS) and Container Service.

Apsara File Storage for HDFS allows you to manage and access data in the same way as the open source Hadoop Distributed File System (HDFS). You can use a distributed file system without the need to modify existing big data applications. The distributed file system offers various features such as unlimited capacity, performance expansion, single namespace, multi-party sharing, high reliability, and high availability.

MaxCompute can interact with Apsara File Storage for HDFS for joint computations after you create external tables.

Apsara File Storage for HDFS supports multiple file formats, such as text file, sequence file, RC file, Parquet, and AVRO. The following example shows how MaxCompute accesses and processes data of Apsara File Storage for HDFS. The text file format is used in this example.

17.2.5.2. Data processing for common tables

17.2.5.2.1. Write data to HDFS

17.2.5.2.1.1. Create an external table

This topic describes how to create an external table for an Apsara File Storage for HDFS data source. It also provides an example for reference.

Run the following commands to create an external table after the `textfile_external` script is compiled:

```
set odps.sql.hive.compatible=true;
drop table if exists textfile_external;
CREATE external TABLE if not exists textfile_external
(
  c_int int ,
  c_tinyint tinyint ,
  c_boolean boolean ,
  c_smallint smallint ,
  c_bigint bigint ,
  c_double double ,
  c_float float ,
  c_date date ,
  c_timestamp datetime ,
  c_string string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES ('field.delim','=',')
STORED AS textfile
location "mcfed:dfs://host:port/user/textfile"
-- host must be set to MountPointId.
-- /user/textfile is the file directory. Replace it with the actual file directory.
TBLPROPERTIES(
  "mcfed.fs.dfs.impl"="com.alibaba.dfs.DistributedFileSystem"
);
```

17.2.5.2.1.2. Write and query data

This topic describes how to use external tables to write and query data of Apsara File Storage for HDFS.

After an external table is created, you can use it the same way you use a common data table. You can use `INSERT OVERWRITE`, `INSERT INTO`, and `SELECT` statements to write data and check whether the write operation succeeds. For more information about the usage of these statements, see [Update the data of a table](#) and [SELECT](#).

17.2.5.2.2. Read data from Apsara File Storage for HDFS

This topic describes how to read data from Apsara File Storage for HDFS and provides an example for reference.

Run the following commands to read data from Apsara File Storage for HDFS after you compile the `textfile_external_read` script:

```
set odps.sql.hive.compatible=true;
drop table if exists textfile_external_read;
CREATE external TABLE if not exists textfile_external_read
(
  c_int int ,
  c_tinyint tinyint ,
  c_boolean boolean ,
  c_smallint smallint ,
  c_bigint bigint ,
  c_double double ,
  c_float float ,
  c_date date ,
  c_timestamp datetime ,
  c_string string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES ('field.delim=',')
STORED AS textfile
location "mcfed:dfs://host:port/user/textfile"
-- host must be set to MountPointId.
-- /user/textfile is the file directory. Replace it with the actual file directory.
TBLPROPERTIES (
  "mcfed.fs.dfs.impl"="com.alibaba.dfs.DistributedFileSystem"
);
-- Create an external table.
select * from textfile_external_read;
select count(*) from textfile_external_read;
select a.c_int,a.c_boolean,a.c_string,b.value from textfile_external_read a join dfstest b on a.c_int=b.id;
-- Query and read data from the external table.
```

17.2.5.3. Data processing for partitioned tables

This topic describes how to use external tables to process data of open source Hadoop Distributed File System (HDFS) data sources in partitioned tables. It also provides an example for reference.

Execute the following statements to create an external table and add partitions to the table to process data:

```
set odps.sql.hive.compatible=true;
drop table if exists textfile_partition;
CREATE external TABLE if not exists textfile_partition
(
  id string,
  name string
)
partitioned by (date string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES ('field.delim','=',')
STORED AS textfile
location "mcfed:dfs://host:port/user/partition/textfile/"
-- host in this command indicates MountPointId.
-- user/partition/textfile/ is the file directory. Replace it with the actual file directory.
TBLPROPERTIES(
  "mcfed.fs.dfs.impl"="com.alibaba.dfs.DistributedFileSystem"
);
-- Create an external table.
alter table textfile_partition add partition (date='20190218');
alter table textfile_partition add partition (date='20190219');
-- Add partitions to the external table.
insert into table textfile_partition partition(date='20190218')select '1','cd' from (select count(*)
from textfile_partition)a;
insert into table textfile_partition partition(date='20190219')select '2','gh' from (select count(*)
from textfile_partition)a;
-- Write data to HDFS.
select * from textfile_partition;
select count(*) from textfile_partition;
select a.id,a.name,b.value from textfile_partition a join dfstest b on a.id=b.id;
-- Query and read data from the external table.
```

17.2.6. TDDL data source

17.2.6.1. Overview

This topic describes how to access and process data of Taobao Distributed Data Layer (TDDL) data sources in MaxCompute.

TDDL is a database middleware that is widely used within Alibaba Group. It supports data sharding, read and write splitting, and failovers by encapsulating MySQL databases. In most cases, TDDL can be directly used to access MySQL databases. TDDL also provides the Corona connection mode. Corona is a MySQL proxy that complies with the standard MySQL protocol and can use a Java Database Connectivity (JDBC) driver to establish a connection.

MaxCompute can access MySQL databases of TDDL. Built-in storage handlers encapsulate native APIs provided by Hadoop such as org, apache, hadoop, mapreduce, lib, and db. The JDBC driver of the MySQL database is used for data communication at the underlying layer.

Some examples are provided to demonstrate how MaxCompute accesses and processes data of TDDL data sources.



Notice In terms of create, read, update, and delete (CRUD) operations, only the following operations are supported:

- MaxCompute reads data from the external table that is created for a MySQL database.
- MaxCompute writes data to the external table in append mode.

17.2.6.2. Preparations

This topic describes the preparations that are required before you create an external table to process data of a Taobao Distributed Data Layer (TDDL) data source.

By default, many features are disabled in MaxCompute V2.0. Therefore, you must manually configure the following parameters to use a new framework to process unstructured data:

```
set odps.sql.hive.compatible=true;
-- Set this parameter for all DDL and DML statements that are used on the external tables for TDDL.
```

```
set odps.sql.udf.java.retain.legacy=false;
-- Set this parameter for all DDL and DML statements that are used on the external tables for TDDL.
```

```
set odps.sql.jdbc.splits.num=3;
-- Set the number of instance splits that are used by MaxCompute to read data from the MySQL database. Maximum value: 256. Default value: 1. You must set this parameter for the SELECT operation on external tables for TDDL.
```

```
set odps.sql.jdbc.reducer.num=3;
-- Set the number of concurrent instances that are used by MaxCompute to write data to the MySQL database. Maximum value: 256. Default value: 64. If the number of concurrent instances in the generated execution plan is less than this value, you do not need to change this value. You must set this parameter if you need to perform the INSERT operation on external tables for TDDL.
```

```
set odps.sql.hive.compatible=true;
-- Call an API operation defined by the open source community to obtain and parse the data types of the MySQL database. You must set this parameter for all DDL and DML statements that are used on the external tables for TDDL.
```

```
set odps.sql.type.system.odps2=true;
-- Set this parameter if new data types are involved in SQL statements, such as CREATE, SELECT, and INSERT. The new data types are TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, and BINARY.
```

17.2.6.3. Create an external table

17.2.6.3.1. Syntax

This topic describes the syntax that is used to create an external table for a Taobao Distributed Data Layer (TDDL) data source.

External tables can be used as interfaces between MaxCompute and databases. The method that is used to process unstructured data of MySQL databases in TDDL is similar to the method used to access and process OSS unstructured data. To create an external table, you must execute the CREATE EXTERNAL TABLE statement first. The following example shows the syntax:

```

DROP TABLE [IF EXISTS] <external_table_name>;
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table_name>
(<column_schemas>)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
location 'jdbc:mysql://path_format'
TBLPROPERTIES (
  ...
);

```

Parameters:

- column schema: supports the data types that are listed in the following table.

Data type mapping

MySQL Type Name	MaxCompute Type Name
TINYINT(UNSIGNED)	TINYINT
SMALLINT(UNSIGNED)	SMALLINT
INT(UNSIGNED)	INT
BIGINT(UNSIGNED)	BIGINT
BOOLEAN	BOOLEAN
FLOAT	FLOAT
DOUBLE	DOUBLE
VARCHAR	VARCHAR
TEXT	STRING
DATE	DATE
DATETIME	DATETIME
DECIMAL	DECIMAL(x, y) (The default precision is (10, 0). An error is returned when an overflow occurs.)

 **Notice** UNSIGNED data types are not supported in MaxCompute. Therefore, loss of precision may occur if UNSIGNED data types are specified. Handle this issue based on scenarios:

- `setproject odps.sql.udf.strict.mode=true;` (strict mode, which is the default mode)
 - Read data from an external table: If loss of precision does not occur when data of the UNSIGNED data type is converted into the SIGNED data type, MaxCompute reads and converts data normally. If loss of precision occurs during data type conversion, the RuntimeException ("value out of range") error is returned.
 - Write data into an external table: MaxCompute does not check data types. You can enable the MySQL database to produce the expected behavior by setting SQL_Mode of the MySQL database. For more information about the setting and behavior of SQL_Mode, see [Server SQL Modes](#).
- `setproject odps.sql.udf.strict.mode=false;` (non-strict mode, which needs to be explicitly configured)
 - Read data from an external table: If loss of precision does not occur when data of the UNSIGNED data type is converted into the SIGNED data type, MaxCompute reads and converts data normally. If loss of precision occurs during data type conversion, NULL is returned.
 - Write data to an external table: MaxCompute does not check data types. You can enable the MySQL database to produce the expected behavior by setting SQL_Mode of the MySQL database.

- **STORED BY:** Only built-in storage handlers are supported. Table field types of TDDL must be the same as the data types that are specified by column schema.
- **LOCATION:** The following three methods can be used to connect to a location:

Method 1: Connect to the MySQL database LOCATION by using a Java Database Connectivity (JDBC) connection string.

```
jdbc:mysql://<user>:<password>@<host>/<databaseName>?useSSL=false&table=<tableName>
```

user is the username of the JDBC connection string that is used to connect to the MySQL database. password is the password of the JDBC connection string used to connect to the MySQL database. host is the network address of the MySQL database instance. databaseName is the name of the MySQL database. tableName is the name of the MySQL database table that corresponds to the external table.

Method 2: Connect to the database LOCATION of TDDL in Corona mode.

```
jdbc:mysql://<user>:<password>@<host>/<databaseName>?useSSL=false&table=<tableName>
```

Method 3: Connect to the database LOCATION of TDDL based on appname.

```
jdbc:mysql://dummy_host?table=<tableName>
```

tableName is the name of the MySQL table that corresponds to the external table. You must specify `odps federation.jdbc.tddl.appname` in the TBLPROPERTIES clause.

 **Notice**

In method 1, MaxCompute interacts with the database by using a direct JDBC connection. You must write the username and password in plaintext. This poses a security risk. Usernames and passwords are hidden when Logview or DESC EXTENDED TABLE is used in MaxCompute. For security concerns, we recommend that you use a separate DDL statement to create an external table before you use the external table.

For example, a project member with higher permissions can create an external table in MaxCompute. Then, other project members can directly use the external table. This prevents project members with lower permissions from using the plaintext username and password. In this case, you do not need to include the plaintext password in SQL scripts.

- **TBLPROPERTIES**: includes the following items.
 - `odps.federation.jdbc.condition`: specifies the filter condition that is used when MaxCompute reads data from the MySQL database. Difference between `odps.federation.jdbc.condition` and `select * from text_test_jdbc_write_external where condition` :

For example, the MySQL database table contains 100 rows of data and you want to obtain 10 data records that meet the specified condition. If you run `odps.federation.jdbc.condition`, data records of the MySQL database table is filtered and MaxCompute reads only 10 data records from the external table. If you run `select * from text_test_jdbc_write_external where condition`, MaxCompute reads 100 data records from the MySQL database table and then obtains 10 data records.

- `odps.federation.jdbc.colmapping`: specifies the column name mapping. Example:

```
-- mysql schema: mysqlId int
-- MaxCompute create table
CREATE EXTERNAL TABLE if not exists table_external
(
  odpsId1 int,
  odpsId2 int
)
STORED BY ...
location ...
TBLPROPERTIES ('odps.federation.jdbc.colmapping'='odpsId1:mysqlId, odpsId2:mysqlId');
```

- `odps.federation.jdbc.insert.type`: specifies the insertion type when data is written to the MySQL database. Only the following types are supported: `SimpleInsert`, `InsertOnDuplicateKeyUpdate`, and `ReplaceInto`. If you do not set this parameter, the default value is `SimpleInsert`.

The `INSERT` statement that is executed in MaxCompute is parsed into the following SQL statements to update data in the database:

```
insert into sqlTable xxx values xxx;
insert into sqlTable xxx values xxx on duplicate key update col1=values(col1), col2=values(col2)
;
replace into sqlTable xxx values xxx;
```

- `odps.federation.jdbc.tddl.app.access.key`: the AccessKey ID for the authorized application.
- `odps.federation.jdbc.tddl.app.secret.key`: the AccessKey secret for the authorized application.
- `odps.federation.jdbc.tddl.appname`: the application name of TDDL. Note that if you specify this parameter, MaxCompute uses the application name to access the MySQL database by using the TDDL SDK.

17.2.6.3.2. Example

This topic provides an example of how to create an external table for a Taobao Distributed Data Layer (TDDL) data source.

The following example shows how to connect to the database of TDDL based on `appname`. In this example, the application name is `ODPS_TDDL_TEST_APP` and the table name is `odps_federation_localrun_write`.

Example:

```
drop table if exists text_test_jdbc_external;
CREATE EXTERNAL TABLE if not exists text_test_jdbc_external
(
  colmapping tinyint, --c_tinyint tinyint,
  c_smallint smallint,
  c_int int,
  c_bigint bigint,
  c_utinyint tinyint,
  c_usmallint smallint,
  c_uint int,
  c_ubigint bigint,
  c_boolean tinyint,
  --c_float float, -- in tddl, not recommend float and double type as it may lost precision
  --c_double double,
  c_string string,
  c_datetime datetime,
  c_decimal decimal
)
STORED BY 'com.aliyun.odps.jdbc.JdbcStorageHandler'
location 'jdbc:mysql://dummy_host?table=odps_federation_localrun_write'
TBLPROPERTIES (
  'odps.federation.jdbc.insert.type'='simpleInsert',
  'odps.federation.jdbc.condition'='c_boolean = 1 and c_int is not null and c_utinyint=127',
  'odps.federation.jdbc.colmapping'='colmapping:c_tinyint',
  'odps.federation.jdbc.tddl.appname'='ODPS_TDDL_TEST_APP',
  'odps.federation.jdbc.tddl.app.access.key'='your tddl app access key',
  'odps.federation.jdbc.tddl.app.secret.key'='your tddl app secret key');
```

17.2.6.4. Read data from an external table

This topic describes how to read data from an external table and provides an example for reference.

Before you perform complex operations such as GROUP JOIN, we recommend that you import data from external tables to MaxCompute tables. This improves the efficiency of data computations. The following example shows how to import data from an associated MySQL external table to MaxCompute.

Create a MaxCompute table

Example:

```
CREATE TABLE if not exists text_test_jdbc_max_compute
(
  c_tinyint tinyint,
  c_smallint smallint,
  c_int int,
  c_bigint bigint,
  c_utinyint tinyint,
  c_usmallint smallint,
  c_uint int,
  c_ubigint bigint,
  c_boolean tinyint,
  --c_float float,
  --c_double double,
  c_string string,
  c_datetime datetime,
  c_decimal decimal
);
```

Import data to a MaxCompute table

Example:

```
insert OVERWRITE TABLE text_test_jdbc_odps select * from text_test_jdbc_read_external;
```

Relationship between creating an external table and importing data to a MaxCompute table

When you create an external table, a data channel is established between MaxCompute and a MySQL database. MaxCompute does not store data in the MySQL database. If data of the external table is missing from the MySQL database, the data is unavailable in MaxCompute.

If data of the MySQL database is imported to a MaxCompute table, the data is stored in MaxCompute. If data is missing from the MySQL database, you can retrieve the data from the MaxCompute table to which the data is imported.

17.2.6.5. Write data to an external table

This topic describes how to write data to an external table.

The column names and data types of the external table must be consistent with those of the database. This ensures that data is correctly written to the external table. For more information about data check actions when loss of precision occurs during data type conversions, see [Syntax](#).

Run the following command to write data to an external table:

```
insert into table text_test_jdbc_external select * from text_test_jdbc_max_compute;
```

Note For MySQL external tables, `Insert INTO MySQL-External-Table` uses the same syntax as `Insert OVERWRITE MySQL-External-Table`. Both statements are executed to append data to external tables. You can use `odps.federation.jdbc.insert.type` to specify the data insertion type. For more information, see [Syntax](#).

17.3. External data sources

17.3.1. HDFS data source (open-source)

17.3.1.1. Overview

This topic describes how to access and process data of open source Hadoop Distributed File System (HDFS) data sources in MaxCompute.

HDFS is the most widely used storage service in the open source community. Most customers use HDFS at the underlying layer of their self-managed big data systems.

MaxCompute uses external tables to access open source HDFS data to facilitate data migration and interact with self-managed customer systems. This reduces customer efforts and costs.

HDFS supports multiple file formats, such as text file, sequence file, RC file, Parquet, and AVRO. Some examples are provided to demonstrate how MaxCompute accesses and processes data of open source HDFS data sources. The text file format is used in the examples.

17.3.1.2. Write data to HDFS

17.3.1.2.1. Create an external table

This topic describes how to create an external table for an open source Hadoop Distributed File System (HDFS) data source. It also provides an example for reference.

After you compile the `textfiletest` script, execute the following statements to create an external table:

```
set odps.sql.hive.compatible=true;
drop table if exists textfiletest;
CREATE EXTERNAL TABLE if not exists textfiletest
(
  c_int int ,
  c_tinyint tinyint ,
  c_boolean boolean ,
  c_smallint smallint ,
  c_bigint bigint ,
  c_double double ,
  c_float float ,
  --c_time datetime ,
  c_date date ,
  c_timestamp datetime ,
  c_string string
)
STORED as TEXTFILE
location 'hdfs://host:port/user/wbyy/';
-- The file directory /user/wbyy/ is for reference only. Replace it with the actual directory.
```

17.3.1.2.2. Write and query data

This topic describes how to use external tables to write and query data from the open source Hadoop Distributed File System (HDFS).

After an external table is created, you can use it the same way that you use a standard table. You can use the `INSERT OVERWRITE`, `INSERT INTO`, and `SELECT` statements to write data and check whether the write operation succeeds. For more information about how to use these statements, see [Update the data of a table](#) and [SELECT](#).

17.3.1.3. Read data from HDFS

This topic describes how to read data from open source Hadoop Distributed File System (HDFS) and provides an example for reference.

After you compile the `testfile_read` script, execute the following statements to read data from open source HDFS:

```
set odps.sql.hive.compatible=true;
drop table if exists testfile_read;
CREATE EXTERNAL TABLE if not exists testfile_read
(
  c_int int ,
  c_tinyint tinyint ,
  c_boolean boolean ,
  c_smallint smallint ,
  c_bigint bigint ,
  c_double double ,
  c_float float ,
  --c_time datetime ,
  c_date date ,
  c_timestamp datetime ,
  c_string string
)
STORED as TEXTFILE
location 'hdfs://host:port/user/wbyy/';
-- The file directory /user/wbyy/ is for reference only. Replace it with the actual file directory.
-- Create an external table.
select * from testfile_read;
-- Query and read data from the external table.
```

17.3.2. MongoDB data source

17.3.2.1. Overview

This topic describes how to access and process data of ApsaraDB for MongoDB data sources in MaxCompute.

ApsaraDB for MongoDB is a stable, reliable, and auto-scaling database service that is fully compatible with MongoDB protocols. ApsaraDB for MongoDB offers a full range of database solutions, such as disaster recovery, backup, restoration, monitoring, and alerting.

MaxCompute can interact with ApsaraDB for MongoDB for joint computations after you create external tables.

Some examples are provided to demonstrate how MaxCompute accesses and processes data of ApsaraDB for MongoDB data sources.

17.3.2.2. Preparations

This topic describes the preparations that are required to create an external table and process data of an ApsaraDB for MongoDB data source.

Before you process data of an ApsaraDB for MongoDB data source, you must deploy ApsaraDB for MongoDB. Example:

1. Run the following command to activate ApsaraDB for MongoDB:

```
bin/mongod --dbpath=./db
```

2. Run the following command to go to the ApsaraDB for MongoDB client:

```
bin/mongo --host=${host}
```

3. Run the following command to create a database:

```
use mongodb
```

4. Run the following command to create a username and password:

```
db.createUser({user: '${user}', pwd: '${password}', roles: [{role:'readWrite',db:'mongodb'}]})
```

5. Run the following command to check whether the user is created. If 1 is returned, the user is created.

```
db.auth('${user}', '${password}')
```

17.3.2.3. Write data to MongoDB

17.3.2.3.1. Create an external table

This topic describes how to create external tables for ApsaraDB for MongoDB data sources and provides an example for reference.

Run the following command to create a collection in ApsaraDB for MongoDB:

```
db.createCollection("${tablename}", { capped : true, autoIndexId : true, size : 6142800, max : 10000 } )
-- The values of the size and max parameters are for reference only. Replace them with the actual values.
```

After the collection is created, execute the following statements to create an external table:

```
set odps.sql.hive.compatible=true;
drop table if exists mongo_table_external;
CREATE external TABLE if not exists mongo_table_external
(
  id string,
  name string
)
STORED BY 'com.mongodb.hadoop.hive.MongoStorageHandler'
location "mcfed:mongodb://${user}:${password}@host:port/mongodb.${tablename}"
TBLPROPERTIES (
  "mcfed.mongo.input.split_size"="2",
  -- The value of input.split_size is for reference only. Replace it with the actual value.
  "mcfed.location"="mongodb://${user}:${password}@host:port/mongodb.${tablename}",
  "mcfed.mongo.input.uri"="mongodb://${user}:${password}@host:port/mongodb.${tablename}",
  "mcfed.mongo.output.uri"="mongodb://${user}:${password}@host:port/mongodb.${tablename}"
);
```

17.3.2.3.2. Write and query data

This topic describes how to use external tables to write and query ApsaraDB for MongoDB data.

After an external table is created, you can use it the same way you use a standard table. You can execute the INSERT OVERWRITE, INSERT INTO, and SELECT statements to write data and check whether the write operation succeeded. For more information about the usage of these statements, see [Update the data of a table](#) and [SELECT](#).

17.3.2.4. Read data from ApsaraDB for MongoDB

This topic describes how to read data from ApsaraDB for MongoDB and provides an example for reference.

After a row of data is inserted into an existing dataset, run the following command to read data from ApsaraDB for MongoDB:

```
set odps.sql.hive.compatible=true;
drop table if exists mongo_read_external;
CREATE external TABLE if not mongo_read_external
(
  id string,
  name string
)
STORED BY 'com.mongodb.hadoop.hive.MongoStorageHandler'
location "mcfed:mongodb://${user}:${password}@host:port/mongodb.${tablename}"
TBLPROPERTIES (
  "mcfed.mongo.input.split_size"="2",
  -- The value of input.split_size is for reference only. Replace it with the actual value.
  "mcfed.location"="mongodb://${user}:${password}@host:port/mongodb.${tablename}",
  "mcfed.mongo.input.uri"="mongodb://${user}:${password}@host:port/mongodb.${tablename}"
);
-- Create an external table.
select * from mongo_external;
-- Query and read data from the external table.
```

17.3.3. HBase data source

17.3.3.1. Overview

This topic describes how to access and process data of ApsaraDB for HBase data sources in MaxCompute.

ApsaraDB for HBase is a distributed database based on Hadoop. It can store petabytes of big data and be used in scenarios that require high-throughput random read and write operations.

MaxCompute allows you to create external tables to interact with ApsaraDB for HBase for joint computations.

The following topics provide examples to show how MaxCompute accesses and processes data of ApsaraDB for HBase data sources.

17.3.3.2. Write data to HBase

17.3.3.2.1. Write data to ApsaraDB for HBase by using an online API

17.3.3.2.1.1. Create an external table

This topic describes how to create an external table of ApsaraDB for HBase data sources. It also provides an example for reference.

Run the following command to create an external table:

```
set odps.sql.hive.compatible=true;
CREATE EXTERNAL TABLE if not exists hbase_table_external
(
  id string,
  cfa string
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('mcfed.hbase.table.name'='${table.name}', 'mcfed.hbase.columns.mapping'=':key,cf:a')
-- cf is the column family in the external table of ApsaraDB for HBase.
location 'hbase://host:port'
TBLPROPERTIES ('hbase.table.name'='${table.name}', 'hbase.columns.mapping'=':key,cf:a', 'mcfed.zookeeper.session.timeout'='30', 'mcfed.hbase.client.retries.number'='1', "mcfed.hbase.zookeeper.quorum"="${host}", "mcfed.hbase.zookeeper.property.clientPort"="${port}");
-- The values of the zookeeper.session.timeout and hbase.client.retries.number parameters are for reference only. Replace them with actual values.
```

17.3.3.2.1.2. Write and query data

This topic describes how to use external tables to write and query ApsaraDB for HBase data.

After an external table is created, you can use it the same way that you use a standard table. You can use the INSERT OVERWRITE, INSERT INTO, and SELECT statements to write data and check whether the write operation succeeds. For more information about the usage of these statements, see [Update the data of a table](#) and [SELECT](#).

17.3.3.2.2. Write data to ApsaraDB for HBase by using bulk load

This topic describes how to write data to ApsaraDB for HBase by using the bulk load method. It also provides an example for reference.

You can use bulk load to import large amounts of data to ApsaraDB for HBase. In the bulk load process, HFiles of ApsaraDB for HBase are generated and then moved to the directory of ApsaraDB for HBase. Compared with the method of calling the put method of ApsaraDB for HBase to write data, the bulk load method provides higher processing efficiency and has less impact on the running of ApsaraDB for HBase. This is because this method bypasses region servers and write-ahead logging (WAL).

MaxCompute performs the following steps to write data to an external table of ApsaraDB for HBase: 1. Sort data in the source table based on row keys. 2. Write data in the HFile format to Hadoop Distributed File System (HDFS). 3. Move HFiles to the directory of ApsaraDB for HBase and notify region servers to load the data.

Create an external table

Run the following command to create an external table:

```
set odps.sql.hive.compatible=true;
CREATE EXTERNAL TABLE if not exists test_hfile_hbase_external
(
  id string,
  cfa string,
  cfb string
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('mcfed.hbase.table.name'='${table.name}', 'mcfed.hbase.columns.mapping'=':key,cf:a,cf:b')
-- cf is the column family in the external table of ApsaraDB for HBase.
location 'hbase://host:port'
TBLPROPERTIES ('hbase.table.name'='${table.name}', 'hbase.columns.mapping'=':key,cf:a,cf:b', 'hfile.tmp.path'='hdfs://hbase-cluster/hfiletmp', 'hadoop.user.name'='admin', 'mcfed.fs.defaultFS'='hdfs://hbase-cluster', 'mcfed.dfs.nameservices'='hbase-cluster', 'mcfed.dfs.ha.namenodes.hbase-cluster'='nn1,nn2', 'mcfed.dfs.namenode.rpc-address.hbase-cluster.nn1'='j63g09343.sqa.eu95:8020', 'mcfed.dfs.namenode.rpc-address.hbase-cluster.nn2'='j63g09355.sqa.eu95:8020', 'mcfed.dfs.client.failover.proxy.provider.hbase-cluster'='org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider', 'mcfed.zookeeper.session.timeout'='30', 'mcfed.hbase.client.retries.number'='1', "mcfed.hbase.zookeeper.quorum"="`${host}`, "mcfed.hbase.zookeeper.property.clientPort"="`${port}`");
-- The values of the zookeeper.session.timeout and hbase.client.retries.number parameters are for reference only. Replace them with actual values.
```

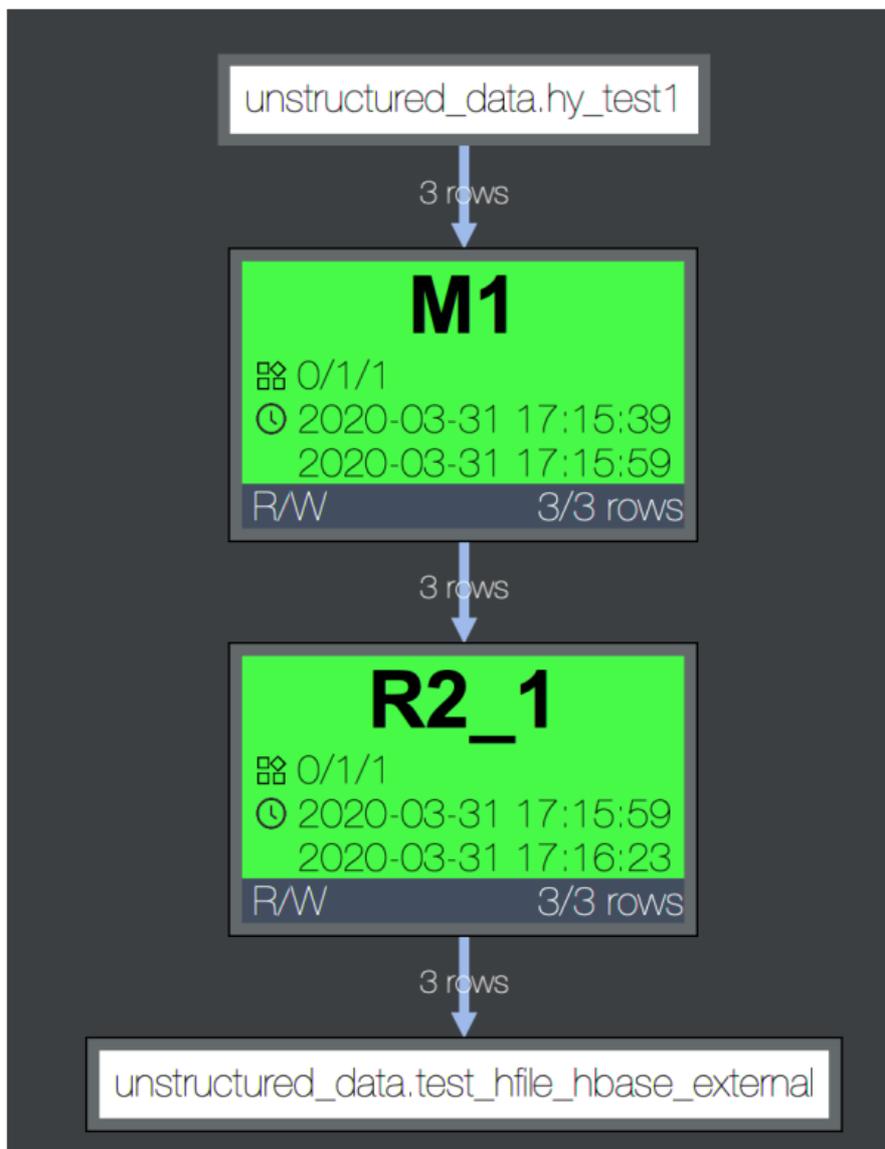
Sort data in the source table based on row keys

The range clustering mechanism of MaxCompute is used to sort data in the source table. If the optimizer generates an execution plan and finds that the table is an external table of ApsaraDB for HBase, the optimizer considers the table as a range-clustered table and sorts all data in the source table.

Example:

```
insert overwrite table test_hfile_hbase_external select uuid(), a_name from hy_test1;
```

The following figure shows the execution plan. Data in the source table `hy_test1` is sorted based on row keys. This step is M1 in the execution plan, as shown in the following figure.



Write data in the HFile format to HDFS

The INSERT statement is executed to automatically write data in the HFile format to HDFS. Each Fuxi instance corresponds to a region for the external table of ApsaraDB for HBase. This step is R2_1 in the execution plan, as shown in the preceding figure. In this step, the sorted data is written in the HFile format to HDFS. You must configure a temporary directory of HDFS to store HFiles to which data is written.

Limits on writing data to HFiles:

- The first column must be a row key, which is `:key` in `hbase.columns.mapping`.
- The other columns must belong to the same column family. Each instance has only one HFile.

Move HFiles to the directory of ApsaraDB for HBase and notify region servers to load the data.

After data in the HFiles of all regions for the external table of ApsaraDB for HBase is written to HDFS, you can perform bulk load. In this step, MaxCompute automatically moves HFiles to the directory of ApsaraDB for HBase and notifies region servers to load the data .

You can run the following command to check the data import result:

```
select * from test_hfile_hbase_external;
```

17.3.3.3. Read data from ApsaraDB for HBase

This topic describes how to read data from ApsaraDB for HBase and provides an example for reference.

After you create a table on an ApsaraDB for HBase client and insert data into the table, run the following commands to read data from the ApsaraDB for HBase table:

```
set odps.sql.hive.compatible=true;
drop table if exists hbase_read_external;
CREATE EXTERNAL TABLE if not exists hbase_read_external
(
  id string,
  name string,
  a string
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('mcfed.hbase.table.name'='${table.name}', 'mcfed.hbase.columns.mapping'=':key,fl:name,fl:a')
-- fl is the column family in the ApsaraDB for HBase table.
location 'hbase://host:port'
TBLPROPERTIES('hbase.table.name'='${table.name}', 'hbase.columns.mapping'=':key,fl:name,fl:a', 'mcfed.zookeeper.session.timeout'='30', 'mcfed.hbase.client.retries.number'='1', "mcfed.hbase.zookeeper.quorum"="${host}", "mcfed.hbase.zookeeper.property.clientPort"="${port}");
-- The values of the zookeeper.session.timeout and hbase.client.retries.number parameters are for reference only. Replace them with actual values.
-- Create an external table.
select * from hbase_read_external;
select count(*) from hbase_read_external;
select a.id,a.name from hbase_read_external a join hbase_test b on a.id=b.id;
-- Query and read data from the external table.
```

18. Unstructured data access and processing (inside MaxCompute)

18.1. Overview

This topic describes how to access and process unstructured volume data by using external tables.

If MaxCompute stores unstructured data as volumes, the data cannot be processed in MaxCompute. You must export the data to an external system for processing.

To address this issue, MaxCompute uses external tables to access various data types. MaxCompute uses external tables to read and write data volumes and processes unstructured data from external data sources, such as Object Storage Service (OSS).

The following topics describe how to create and access volume external tables.

18.2. Create a volume external table

18.2.1. Syntax

This topic describes the syntax that is used to create a volume external table.

You can first execute the CREATE EXTERNAL TABLE statement to create an external table. The following example shows the syntax:

```
DROP TABLE [IF EXISTS] <external_table_name>;
Create External Table [IF NOT EXISTS] <external_table_name>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED BY '<StorageHandler>'
[WITH SERDEPROPERTIES (
    'name'='value'
)]
LOCATION 'volume://...'
[USING '<ResourceName>']
;
```

Parameters:

- **STORED BY:** Two built-in storage handlers `com.aliyun.odps.CsvStorageHandler` and `com.aliyun.odps.TsvStorageHandler` are supported. They can be used to read and write CSV files where the column delimiter is a comma and the row delimiter is `\n` or TSV files where the column delimiter is `\t` and the row delimiter is `\n`). If the built-in storage handlers cannot be used, you can build a custom storage handler.
- **WITH SERDEPROPERTIES:** specifies table attributes such as delimiters for a custom storage handler.
- **LOCATION:** the location format of the table.

Format:

```
volume://[project_name]/volume_name/partition_value
```

Example:

```
volume://test_project/volume_data/20190102
```

project_name is optional. If project_name is not specified, the current project is used to obtain volume data after the DML SQL statement is executed. In the preceding example, if the current project is myproject, you can use the following location format:

```
volume:///volume_data/20190102
```

Notice

- The location of a non-partitioned table must point to a volume partition, instead of the volume.
- The location of a partitioned table must point to the volume.
- The volume path cannot contain an equal sign (=) and does not support the default standard partition path `ds=2017071` that is used when a partition is created. The partition path must be customized. The custom partition path can be any path supported by the volume. For example, if the partition path is 20190102, the path combined with volume path can be `volume:///test_project/volume_data/20190102`.

- USING: specifies a storage handler. To use a custom storage handler, you must export the custom storage handler as a JAR package and then add it to MaxCompute as a JAR resource.

18.2.2. Use a built-in storage handler to create a table

This topic describes how to use a built-in storage handler to create a table.

You can use a built-in storage handler to create a partitioned table or non-partitioned table.

Create a non-partitioned table

Example:

```
DROP TABLE IF EXISTS volume_ext;
Create External Table volume_ext
(
  key string,
  value string
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'--This is the built-in storage handler.
LOCATION 'volume:///test_project/volume_data/20190102'
;
```

Create a partitioned table

Example:

```
DROP TABLE IF EXISTS volume_ext_pt;
Create External Table volume_ext_pt
(
  key string,
  value string
)
PARTITIONED BY (ds string)
STORED BY 'com.aliyun.odps.CsvStorageHandler'--This is the built-in storage handler.
LOCATION 'volume://test_project/volume_data'
;
ALTER TABLE volume_ext_pt DROP IF EXISTS PARTITION (ds="20190102");
ALTER TABLE volume_ext_pt ADD PARTITION (ds="20190102") LOCATION "volume://test_project/volume_data/20190102";
```

18.2.3. Use a custom StorageHandler to create a table

This topic describes how to use a custom StorageHandler to create a table.

If the built-in StorageHandlers are unable to meet your business requirements, you can customize a StorageHandler by using Java and specify some attributes of the Volume external table through which you want to process data.

The following example shows how to create a table:

Assume that the data type of the table that you want to create is TXT and the column delimiter is |. You can perform the following steps to create an external table:

1. Use the MaxCompute Studio or MaxCompute Eclipse development tool to customize Java classes.
2. Export the JAR package. In this example, the package name is odps-volume-example.jar.
3. Run the following command to add the JAR package to MaxCompute as a resource:

```
add jar odps-volume-example.jar -f;
```

4. Run the following commands to create an external table:

```
DROP TABLE IF EXISTS volume_ext;
Create External Table volume_ext
(
  key string,
  value string
)
STORED BY 'com.aliyun.odps.udf.example.text.TextStorageHandler'
WITH SERDEPROPERTIES (
  'delimiter'='|'
)
LOCATION 'volume://myproject/volume_data/20190102'
USING 'odps-volume-example.jar'
;
```

 **Note** After the external table is created, you can process the volume data by using the external table.

18.3. Access a Volume external table

This topic describes how to access a Volume external table.

Volume external tables can be accessed in the same way that you access a MaxCompute table. Example:

```
select key,value from volume_ext_pt where ds="20190102";
```

19.Intelligent storage enhancement package

19.1. Backup and restoration

During data development, if you accidentally delete data, you need to restore the data. If a data issue occurs after you execute the `INSERT INTO` or `INSERT OVERWRITE` statement, you need to restore the data to a previous version. This topic describes how to back up and restore data in MaxCompute.

Enable the backup and restoration feature

Prerequisites

The changelog feature is disabled. You can run the `desc project <projectname>;` command to check whether the changelog feature is disabled. If the value of `Properties odps.changelogs.ttl` is not 0, you must run the `setproject odps.changelogs.ttl=0;` command to disable the changelog feature.

After the changelog feature is disabled, the backup and restoration feature is automatically enabled for MaxCompute. The default data retention period is one day. The project owner or a RAM user that is assigned the Super_Administrator role can run the following command to change the data retention period.

```
setproject odps.timemachine.retention.days=days;
```

days indicates the number of days during which the backup data can be retained. During the retention period, you can restore data of the current version to the backup data of a previous version. Valid values: 0 to 7. The value 0 indicates that the backup and restoration feature is disabled.

Commands that are related to the backup and restoration feature

After the backup and restoration feature is enabled, you can run the following commands to view information about the backup versions of a table or a partition of the table and restore the table or partition to a specific data version.

- Commands to view backup information

Command	Description
<code>show history for tables;</code>	View information about the current table and its backup table. Note To run this command, you must have the List permission on the current project.
<code>show history for table table_name;</code>	View the backup information of a specified table. Note If the table exists, you must have the ShowHistory permission on the table. If the table does not exist, you must have the List permission on the project.

Command	Description
<code>show history for table table_name ('id'='xxxx');</code>	View the backup information of a table that is dropped.
<code>show history for table table_name PARTITION (pt='xxx');</code>	View the backup information of a specified partition.
<code>show history for table table_name PARTITION ('id'='xxxx');</code>	View the backup information of a partition that is dropped.

• Commands to restore a table

Command	Description
<code>restore table table_name ('id'='xxxx');</code>	Restore a table that is dropped.
<code>restore table table_name to LSN 'xxxx';</code>	Restore a table to a specified version.
<code>restore table table_name to LSN 'xxxx' as new_table_name;</code>	Restore a table to a specified version and save the table as a new table.

Note If the table exists, you must have the Update permission on the table. If the table does not exist, you must have the CreateTable permission on the project.

• Commands to restore a partition

Command	Description
<code>restore table table_name PARTITION ('id'='xxxx');</code>	Restore a partition that is dropped. You can restore multiple partitions at the same time.
<code>restore table table_name PARTITION (pt='xxx') [PARTITION (pt='xxx')] to LSN 'xxxx';</code>	Restore a partition to a specified version. You can restore multiple partitions at the same time.
<code>restore table table_name PARTITION (pt='xxx') [PARTITION (pt='xxx')] to LSN 'xxxx' as new_table_name;</code>	Restore a partition to a specified version and save the partition in a new table. You can restore multiple partitions at the same time.

Note You can specify only one version that corresponds to a log sequence number (LSN) in the command for restoring a partition. When you restore multiple partitions at the same time, each partition is restored to the version that corresponds to the specified LSN in the command or the version that corresponds to the LSN before the specified LSN. For example, the versions for pt1 correspond to LSNs 100, 102, 104, and 106 and the versions for pt2 correspond to LSNs 101, 103, 104, and 105. After you run the `restore table table_name PARTITION (pt='1') PARTITION (pt='2') to LSN '102';` command, the pt1 partition is restored to the version that corresponds to LSN 102 and the pt2 partition is restored to the version that corresponds to LSN 101.

Examples

The following three tables are used in the examples:

- orders: a non-partitioned table
- orders_delta: a non-partitioned table

- orders_pt: a partitioned table

Example 1: Restore the orders table that is dropped

1. Drop the orders table. Sample command:

```
drop table orders;
Confirm to "drop table orders" (yes/no)? yes
OK
```

2. View the backup information of the orders table and obtain the ID of the table. Sample command:

```
show history for tables;
```

3. View the backup information of the orders table. Sample command:

```
show history for table orders ('id'='25899d487db04852a7f4331491f8****');
```

4. Restore the orders table. Sample command:

```
restore table orders ('id'='25899d487db04852a7f4331491f8****');
```

Example 2: Restore the orders table whose data is updated

1. Update data in the orders table. Sample command:

```
-- Execute the INSERT INTO statement to update table data. You can also execute the INSERT OVERWRITE statement to update table data.
insert into orders select * from orders_delta;
OK
-- You can also execute the following statement to update table data:
Truncate table orders;
OK
```

2. View the backup information of the orders table and obtain the LSNs. Sample command:

```
show history for table orders;
```

3. Restore data in the orders table. Sample command:

```
-- Restore data in the orders table from the latest version to the version before truncation, and save the table as a new table.
restore table orders to LSN 'lsn_no' as orders_restore;
```

Example 3: Restore a partition that is dropped from the orders_pt table

1. Drop a partition from the orders_pt table. Sample command:

```
alter table orders_pt drop if exists partition(pt='1998-08-02');
```

2. View the backup information of the orders_pt table and obtain the partition ID. Sample command:

```
show history for table orders_pt;
```

3. Restore the partition that is dropped. Sample command:

```
restore table orders_pt PARTITION('id'='88ba032fb9524a75b1daddb3e29b****');
```

Example 4: Restore a partition whose data is updated in the orders_pt table

1. Update data in the orders_pt table. Sample commands:

```
insert into orders_pt(pt='1999-01-01') select * from orders_pt;
OK
insert overwrite orders_pt(pt='1999-01-02') select * from orders_pt;
OK
```

2. View the backup information of the orders_pt table or a partition of the table and obtain the LSNs. Sample commands:

```
-- View the backup information of the orders_pt table, including all partitions of the table.
show history for table orders_pt;
-- View the backup information of a specific partition.
show history for table orders_pt PARTITION(pt='1999-01-01');
show history for table orders_pt PARTITION(pt='1999-01-02');
```

3. Restore partition data. Sample commands:

```
-- Restore partition data to a specified version.
restore table orders_pt PARTITION(pt='1999-01-01') PARTITION(pt='1999-01-02') to LSN '0000000000
000026';
-- Restore the partition to a specified version and save the partition in a new table.
restore table orders_pt PARTITION(pt='1999-01-01') PARTITION(pt='1999-01-02') to LSN '0000000000
000026' as orders_pt_restore;
```

Example 5: Restore the orders_pt table that is dropped

1. Drop the orders_pt table. Sample command:

```
drop table orders_pt;
```

2. View the backup information of the orders_pt table and obtain the ID of the table. Sample command:

```
show history for tables;
```

3. Restore the orders_pt table. Sample command:

```
restore table orders_pt('id'='a26c91705f53483493474d7f28c1****');
```

20. MaxCompute multi-region deployment

20.1. Overview

MaxCompute supports multi-region deployment. This topic describes how to deploy control clusters and compute clusters across regions.

Centralized deployment of control clusters is designed to configure resources and manage computing tasks.

Independent deployment of compute clusters for each region is designed to create projects and deliver computing tasks.

20.2. Features

This topic describes the features of multi-region deployment on MaxCompute.

Multi-region deployment on MaxCompute has the following features:

- A MaxCompute system can manage multiple clusters in different regions.
- Data exchanges between clusters within MaxCompute, and data replication and synchronization between clusters are managed based on configured policies.
- Metadata is stored in a centralized manner. Therefore, the infrastructure requirements, such as the network connections of different data centers, are relatively high.
- A unified account system is used.
- The development systems for big data applications, such as DataWorks, are used for all clusters in all regions.
- MaxCompute must run in multi-cluster mode to support multi-region deployment.

Note The conditions and limits on changing the cluster mode must meet the following requirements:

- The network bandwidth must be sufficient to support multi-region data synchronization and link redundancy.
- Control clusters in the central region have a high latency for basic services, such as Apsara Stack DNS and Tablestore. Therefore, we recommend that you deploy basic services in the same data center to limit the network latency within 5 ms.
- The network latency between control clusters in the central region and compute clusters in other regions is within 20 ms.
- Clocks must be synchronized between clusters in different regions and between servers in the same cluster.
- The network bandwidth must be sufficient to support data replication between clusters.
- Apsara Stack DNS is required.
- Servers in different clusters can communicate with each other, and the clusters have similar network infrastructure (1-Gigabit or 10-Gigabit).

- The O&M and upgrades for multi-region deployment are different from those for single-cluster deployment. Multi-region deployment requires higher on-site O&M capabilities.

20.3. Instructions

This topic describes the multi-region deployment instructions for MaxCompute.

Instructions:

- Computing tasks on MaxCompute are implemented in a cluster based on data distribution within MaxCompute. Cross-cluster replication and direct reads are used for data exchanges between clusters based on the cluster configurations and distribution of data.
- Data replication and synchronization can be performed between clusters by table or partition based on the cluster configurations.
- You must plan the relationship between MaxCompute projects and clusters based on your business requirements. You can select one or more clusters for a single project.

Note If you select multiple clusters for a project, data replication is performed among these clusters.

- If cross-cluster computing cannot be implemented for business in a project, you can distribute the data of this business to different clusters and divide the computing tasks based on data distribution.

Note Cross-cluster data replication and management require management policies and maintenance.

- The bandwidth between clusters in different remote data centers must be sufficient for data replication or read-only operations.
- The two data centers that are involved in cross-region operations share the bandwidth. As a result, cross-cluster replication and read-only operations that involve large amounts of data may consume all of the bandwidth between the two data centers, which affects transmission of other data.
- A cluster failure in the primary data center affects the entire service because the primary data center stores information, including metadata and accounts.
- If the secondary data center fails, projects that have data stored on a cluster of this data center and the O&M of services are affected.

Note If project data is separately stored in the primary data center, the failure of the secondary data center does not affect the computing tasks of the project.

- In common scenarios, data is distributed and replicated based on business characteristics, and computing tasks are performed based on data types in different clusters. The combined results of different MaxCompute tasks are applied.

20.4. Examples of multi-region deployment

20.4.1. Synchronize table data among multiple clusters

This topic provides an example of how to synchronize table data between multiple clusters in a multi-region deployment scenario.

Prerequisites

- When you create a project in the AdminConsole, you have selected two clusters and set one as the default cluster. The following figure shows an example.

Note

- In this example, the project is multiregion, and AT-MDU-TEST and AT-5KN are selected for it, with AT-MDU-TEST as the default cluster.
- The endpoint of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`. 9090 indicates the port number of ODPS AG.
- To perform this step, choose **MaxCompute Configuration > Project Management > Create Project**.

- You have configured cross-cluster replication for the project. The following example shows how to configure it.
 - i. Choose **MaxCompute Configuration > Global Cross-cluster Replication Configuration** to go to the **Global Cross-cluster Replication Configuration** page.
 - ii. In the Add Item part of the **InfoBetweenClusters** section, set key to **AT-5KN#AT-MDU-TEST**, set both **availableBandwith** and **totalBandwith** to **2000**, and click **add**.
 - iii. Choose **MaxCompute Configuration > Project Management**. Find multiregion and click **Cross-cluster Replication Configuration**.
 - iv. In the dialog box that appears, configure the parameters, as shown in the following figure.
 - v. Click **Save** and then click **Start Replication**.

Procedure

Example:

1. In the AG of the default cluster AT-MDU-TEST, construct an upload tunnel data file in the directory of the same level as console, such as `echo "testtest" > uploaddata`.
2. Go to the console command line and run the following commands to create a table and insert data into the table:

```
use multiregion;
create table t1 (s string);
tunnel upload uploaddata t1;
```

3. Run the following command to check whether the data is inserted into the table:

```
select * from t1;
```

4. In the AG of AT-MDU-TEST, view the Apsara Distributed File System directory for this table. Run the following command to check whether data exists in the directory:

```
pu ls product/aliyun/odps/multiregion/data/t1
```

5. In the AG of AT-5KN, view the Apsara Distributed File System directory for this table. Run the following command to check whether data exists in the directory:

```
pu ls /apsara/odps/mdutesting/multiregion/data/t1
```

6. If data is displayed after you perform Steps 4 and 5, cross-cluster data synchronization is successful.

20.4.2. Query the status of data synchronization between primary and secondary clusters

This topic provides an example of how to query the status of data synchronization between the primary and secondary clusters in multi-region deployment scenarios.

Prerequisites

- When you create a project in the AdminConsole, you have selected two clusters and set one as the default cluster. The following figure shows an example.

Note

- In this example, the project is multi-region, and AT-MDU-TEST and AT-5KN are the clusters selected for the project. AT-MDU-TEST is set as the default cluster.
- The endpoint of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`. 9090 indicates the port number of ODPS AG.
- To perform this step, choose **MaxCompute Configuration > Project Management > Create Project**.

- You have configured cross-cluster replication for the project. Detailed operations:
 - i. Choose **MaxCompute Configuration > Global Cross-cluster Replication Configuration** to go to the **Global Cross-cluster Replication Configuration** page.
 - ii. In the Add Item part of the **InfoBetweenClusters** section, set key to **AT-5KN#AT-MDU-TEST**, set availableBandwidth and totalBandwidth to **2000**, and then click **Add**.
 - iii. Choose **MaxCompute Configuration > Project Management**. Find multi-region and click **Cross-cluster Replication Configuration**.
 - iv. In the dialog box that appears, configure the parameters, as shown in the following figure.
 - v. Click **Save** and click **Start Replication**.

Procedure

Example:

1. Create the Bodychecksync configuration file, which is used to send Admin task.

```

<Instance>
  <Job>
    <Comment>
    </Comment>
    <Priority>1</Priority>
    <Tasks>
      <Admin>
        <Name>task_1</Name>
        <Comment>test</Comment>
        <Config>
          <Property>
            <Name>PROJECT</Name>
            <Value>multiregion</Value>
          </Property>
          <Property>
            <Name>CLUSTER</Name>
            <Value>AT-MDU-TEST</Value>
          </Property>
        </Config>
        <Command>GET_UNREPLICATED_OBJECTS</Command>
      </Admin>
    </Tasks>
    <DAG>
      <Comment/>
      <RunMode>Sequence</RunMode>
    </DAG>
  </Job>
</Instance>

```

2. Compile the header.

```
content-type: application/xml
```

3. Run the following command to send Admin task:

```
CLTrelease/bin/odpscmd -e "http post /projects/admin_task_project/instances -header=header -content=bodychecksync;"
```

4. Run the following command to check the instance execution results:

```
CLTrelease/bin/odpscmd --project=admin_task_project -e "wait 20180711050550317gnege3ms2;"
```

5. Access the LogView URL generated in Step 4 in a browser.

6. On the page that appears, click **Detail**. The status of data synchronization between the primary and secondary clusters is displayed.

20.4.3. Cross-region direct read

This topic provides examples of cross-region direct read operations in multi-region deployment scenarios.

Cross-region direct read can be implemented with or without cross-cluster access configured.

 **Note** If cross-cluster access is not configured and the data volume is large, the cross-region direct read takes a long period of time.

Direct read when cross-cluster access is not configured

Example:

1. In the AdminConsole, create two projects: testmdu and testcross5kn.

 **Note**

- In this example, AT-MDU-TEST and AT-5KN are selected for the testmdu project, with AT-MDU-TEST as the default cluster. AT-5KN is selected for the testcross5kn project and serves as the default cluster.
- The URL of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`. 9090 indicates the port number of MaxCompute AG.
- The entry point to create the project is **MaxCompute Configuration > Project Management > Create Project**.

2. Create tables for the testmdu and testcross5kn projects.

```
create table testrep(s string );
create table tablecross5kn (s string );
```

3. Construct some data for the testrep and tablecross5kn tables.
4. Run the following commands to read the table data in the testmdu project directly from the testcross5kn project:

```
use testcross5kn;
select * from testmdu.testrep;
```

 **Note** If the required data is returned, cross-region direct read is successful.

Direct read when cross-cluster access is configured

Example:

1. In the AdminConsole, create two projects: testmdu and testcross5kn.

 **Note**

- In this example, AT-MDU-TEST and AT-5KN are selected for the testmdu project, with AT-MDU-TEST as the default cluster. AT-5KN is selected for the testcross5kn project and serves as the default cluster.
- The URL of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`. 9090 indicates the port number of MaxCompute AG.
- The entry point to create the project is **MaxCompute Configuration > Project Management > Create Project**.

2. Create tables for the testmdu and testcross5kn projects.

```
create table testrep(s string );
create table tablecross5kn (s string );
```

3. Construct some data for the testrep and tablecross5kn tables.
4. Configure cross-cluster replication for the testmdu project.

 **Note** The entry point is **MaxCompute Configuration > Project Management**. Find testmdu and click **Cross-cluster Replication Configuration**.

5. Run the following commands to read the table data in the testmdu project directly from the testcross5kn project:

```
use testcross5kn;
select * from testmdu.testrep;
```

 **Note** If the required data is returned, cross-region direct read is successful.

20.4.4. Cross-region JOIN

This topic provides an example of cross-region JOIN in multi-region deployment scenarios.

Example:

1. In Apsara Stack MaxCompute AdminConsole, create two projects: crossregion and crossregion02.

 **Note**

- o In this example, select the AT-MDU-TEST and AT-70N compute clusters for the crossregion project. The default cluster for the crossregion project is AT-MDU-TEST. Select only the default cluster AT-70N for the crossregion02 project.
- o The endpoint of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`. 9090 indicates the port number of ODPS AG.
- o To perform this step, choose **MaxCompute Configuration > Project Management > Create Project**.

2. Configure cross-cluster replication for the crossregion project.

 **Note** Choose **MaxCompute Configuration > Project Management**. Find the crossregion project and click **Cross-cluster Replication Configuration**.

3. Create the business table for the crossregion project.

```
create table business (bid bigint, name string, phone string, address string, region string);
```

4. Generate data for the business table.
5. Run the following command to import data into the business table:

```
tunnel upload business business;
```

6. Create the product table for the crossregion02 project.

```
create table if not exists product (pid bigint, name string, type string, color string, bid bigint);
```

7. Generate data for the product table.
8. Run the following command to import data into the product table:

```
tunnel upload product product;
```

9. Run the following command in Apsara Stack MaxCompute AdminConsole to obtain specified data from the business and product tables:

```
select pro.pid, pro.name, pro.type, pro.color, bus.name, bus.phone from crossregion02.product pro join crossregion.business bus on pro.bid=bus.bid where bus.region='Shanghai';
```

 **Note** If the specified data is returned, the cross-region JOIN operation is complete.

21. Security solution

21.1. Intended users

This section describes the users for which the MaxCompute security solution is intended.

This MaxCompute security solution is intended for all owners and administrators of MaxCompute projects, and users who are interested in the MaxCompute multi-tenant data security system. The MaxCompute multi-tenant data security system supports the following features:

- User authentication
- User and authorization management of projects
- Cross-project resource sharing
- Project data protection

21.2. Quick start

This topic describes the procedure of the MaxCompute security solution. This provides guidance for you to perform MaxCompute security operations.

Add a user and grant permissions to the user

Scenario: Jack is the administrator of the prj1 project. Alice, who has an Alibaba Cloud account (alice@aliyun.com), joins the prj1 project. Alice applies for the permissions to create and view tables and submit jobs.

Statements:

```
use prj1
add user aliyun$alice@aliyun.com
-- Add a user.
grant List, CreateTable, CreateInstance on project prj1 to user aliyun$alice@aliyun.com
-- Grant permissions to the user.
```

 **Note** Only the project administrator can perform this operation.

Add a role and grant permissions to the role by using ACL-based authorization

Scenario: Jack is the administrator of the prj1 project. Three new members Alice, Bob, and Charlie join the project as data reviewers. The members apply for the permissions to view tables, submit jobs, and read the user profile table.

The project administrator can use ACL-based authorization for objects in this scenario. Statements:

```
use prj1
add user aliyun$alice@aliyun.com
-- Add users.
add user aliyun$bob@aliyun.com
add user aliyun$charlie@aliyun.com
create role tableviewer
-- Create the tableviewer role.
grant List, CreateInstance on project prj1 to role tableviewer
-- Grant permissions to the tableviewer role.
grant Describe, Select on table userprofile to role tableviewer
grant tableviewer to aliyun$alice@aliyun.com
-- Assign the tableviewer role to users.
grant tableviewer to aliyun$bob@aliyun.com
grant tableviewer to aliyun$charlie@aliyun.com
```

Package and share resources

Scenario: Jack is the administrator of the prj1 project. John is the administrator of the prj2 project. Jack wants to share some resources of the prj1 project, such as the datamining.jar file and the sampletable table to the prj2 project owned by John. If Bob in the prj2 project wants to access these resources, John can use ACL- or policy-based authorization to grant permissions to Bob, without the assistance of Jack.

Statements:

1. Jack, the administrator of the prj1 project, creates a resource package in the project.

```
use prj1
create package datamining
-- Create a package.
add resource datamining.jar to package datamining
-- Add resources to the package.
add table sampletable to package datamining
-- Add a table to the package.
allow project prj2 to install package datamining
-- Share the package to the prj2 project.
```

2. John installs the package in the prj2 project.

```
use prj2
install package prj1.datamining
-- Install the package.
describe package prj1.datamining
-- View the resources of the package.
```

3. John grants Bob the permission to use the package.

```
use prj2
grant Read on package prj1.datamining to user aliyun$bob@aliyun.com
-- Use ACL-based authorization to grant Bob the permission to use the package.
```

 **Note** For more information about cross-project resource sharing, see [Cross-project resource sharing](#).

Configure project data protection

Scenario: Jack is the administrator of the prj1 project. This project contains sensitive data, such as user IDs and shopping records. The project also stores a large number of data mining algorithms that are proprietary. Jack wants to protect the sensitive data and algorithms in the project and allows only users in the project to access the data. In this case, the data cannot be transferred to other projects.

Statements:

```
use prj1
set ProjectProtection=true
-- Enable project data protection.
```

If project data protection is enabled, data in the project can be shared only within the project. Data cannot be transferred to other projects. In some cases, for example, Alice wants to export data tables to other projects for business purposes. This operation is approved by the project administrator. MaxCompute provides two methods to export data from a project for which project data protection is enabled.

Method 1: Create an exception policy. For more information, see [Data export methods when project protection is enabled](#).

1. Create a policy file. For example, create the `/tmp/exception_policy.txt` file that only allows Alice to use SQL tasks to export the t1 table from the prj1 project. Content in the `exception_policy.txt` file:

```
{
  "Version": "1",
  "Statement": [{
    "Effect": "Allow",
    "Principal": "ALIYUN$alice@aliyun.com",
    "Action": ["odps:Describe", "odps:Select"],
    "Resource": "acs:odps:*:projects/prj1/tables/t1",
    "Condition": {
      "StringEquals": { "odps:TaskType": "SQL"
    }
  }
}]
```

2. Method 1: Configure an exception policy.

```
use prj1
-- Enable project data protection and configure an exception policy.
set ProjectProtection=true with exception /tmp/exception_policy.txt
```

Note To configure an exception policy, make sure that the principal is not granted the DROP and CREATE permissions or the UPDATE permission on resources. This prevents data leaks caused by time-of-check to time-of-use (TOCTOU).

Method 2: Configure trusted projects. Configure the prj2 project as a trusted project of the prj1 project to enable data transfer from the prj1 project to the prj2 project. For more information, see [Data export methods when project protection is enabled](#).

```
use prj1
add trustedproject prj2
```

Note The package-based resource sharing and project data protection mechanisms are two orthogonal security mechanisms.

In MaxCompute, resource sharing has a higher priority than project data protection. If an object in a protected project is shared with other projects by using the package mechanism, cross-project access to this object is not subject to the rules for project data protection.

21.3. User authentication

This topic describes user authentication for MaxCompute security solutions.

User authentication is used to verify the identity of a request sender. Authentication typically includes:

- Verifying the identity of a message sender
- Checking whether the message was tampered with before it is received.

21.4. Project user and authorization management

21.4.1. Overview

This topic describes user management, role management, and authorization for projects.

Projects are the foundation of the MaxCompute multi-tenant system and the basic unit of data management and computing. If you create a project, you are the owner of this project. You can manage all objects in the project, such as tables, instances, resources, and user-defined functions (UDFs). Users, except the project owner, can access objects in the project only after these users obtain approvals from the project owner.

For example, if Alice is the owner of the test_project project, and another user from the project team of Alice requests to access the resources in the test_project project, Alice can use the methods described in this topic to perform user management and authorization. If the user is not from the project team of Alice, we recommend that Alice implement cross-project resource sharing. For more information, see [Cross-project resource sharing](#).

21.4.2. User management

This topic describes how to manage users in projects.

Add a user

If the project owner, Alice, decides to authorize another user, Alice must add the user to this project first. Alice can authorize only the users in the project.

Run the following command to add a user:

```
add user <full_username>
-- Add a user to a project.
```

Remove a user

If a user leaves the project team, Alice must remove the user from the project. After a user is removed from the project, the user no longer has permissions to access resources in the project.

Run the following command to remove a user from a project:

```
remove user <full_username>
-- Remove a user from a project.
```

Note

- After a user is removed, the user no longer has permissions to access resources in the project.
- Before you remove a user who has been assigned a role, you must first revoke the role. For information about roles, see [Role management](#).
- After a user is removed, the ACL-based authorization related to the user is retained. The policy-based authorization at the role level is revoked, but the policy-based authorization at the project level is retained. If the user is added to the project again, the historical permissions granted by using ACL-based authorization are activated again.
- Removed users may be added to the original projects as different identities. This poses data security risks. If you are a project owner or use an account that has been assigned the Admin or Super_administrator role, you can run the `purge privs from user <username>;` command to permanently delete the permission information of removed users. The permission information includes the permissions granted by using ACL-, label-, and policy-based authorization information. If you do not remove the relevant users from the project, the following error is returned after you run the `purge privs from user <username>;` command: `Principal still exist in the project`

21.4.3. Role management

This topic describes the operations that you can perform to manage roles in projects.

A role is a collection of access permissions. A role can be used to assign the same permissions to a group of users. Role-based authorization can greatly simplify the authorization process and reduce authorization management costs. We recommend that you use role-based authorization to authorize users.

An admin role is automatically created when a project is created. This role is granted permissions to access all objects of the project, manage users and roles, and authorize users and roles. Compared with the project owner, the user who is assigned the admin role cannot assign another user with the admin role, configure security rules for a project, or change the authentication model of the project. Permissions of the admin role cannot be modified.

Commands that are used for role management:

```
create role <rolename>
-- Create a role.
drop role <rolename>
-- Delete a role.
grant <rolename> to <username>
-- Assign a role to a user.
revoke <rolename> from <username>
-- Revoke a role from a user.
```

Note When you delete a role, MaxCompute checks whether users who are assigned this role exist. If these users exist, the role fails to be deleted. To delete the role, you must revoke this role from all users.

21.4.4. ACL-based authorization actions

This topic describes the actions of ACL-based authorization for projects.

Authorization usually involves three elements: subject, object, and action. In a MaxCompute project, a subject is a user and various types of objects are included. Different types of objects support different actions.

The following table describes the types of objects and actions supported by these objects in a MaxCompute object.

Types and supported actions of objects

Object	Action	Description
Project	Read	Views project information (excluding project objects), such as the creation time.
Project	Write	Updates project information (excluding project objects), such as comments.
Project	List	Views objects of all types in a project.
Project	CreateTable	Creates tables in a project.
Project	CreateInstance	Creates instances in a project.
Project	CreateFunction	Creates functions in a project.
Project	CreateResource	Creates resources in a project.
Project	CreateJob	Creates jobs in a project.
Project	CreateVolume	Creates volumes in a project.
Project	All	Supports all the preceding actions on projects.
Table	Describe	Reads the metadata of a table.
Table	Select	Reads data of a table.
Table	Alter	Modifies the metadata of a table. Adds or deletes partitions.
Table	Update	Overwrites or adds data of a table.
Table	Drop	Deletes a table.
Table	All	Supports all the preceding actions on tables.
Function	Read	Reads data and executes functions.
Function	Write	Updates data.
Function	Delete	Deletes data.
Function	All	Supports all the preceding actions on functions.
Resource,Instance,Job,Volume	Read	Reads data of resources, instances, jobs, and volumes.
Resource,Instance,Job,Volume	Write	Updates resources, instances, jobs, and volumes.
Resource,Instance,Job,Volume	Delete	Deletes resources, instances, jobs, and volumes.

Object	Action	Description
Resource, Instance, Job, Volume	All	Supports all the preceding actions on resources, instances, jobs, and volumes.

Note In the preceding table, the CREATE TABLE permission on projects and the SELECT, ALTER, UPDATE, and DROP permissions on tables must be used with the CREATE INSTANCE permission on projects. If these permissions are not used with the CREATE INSTANCE permission, these permissions may become ineffective.

After you add users or create roles, you must grant permissions to these users or roles. ACL-based authorization of MaxCompute is an object-based authorization. An access control list (ACL) that contains the authorization data is considered a resource of an object. Authorization can be performed only when the object exists. If the object is deleted, the ACL is automatically deleted.

ACL-based authorization of MaxCompute is performed by running the GRANT or REVOKE command defined in SQL-92. These commands are used to grant or revoke permissions to or from objects in a project.

Command syntax:

```
grant actions on object to subject
revoke actions on object from subject
actions ::= action_item1, action_item2, ...
object ::= project project_name | table schema_name | instance inst_name | function func_name | resource res_name
subject ::= user full_username | role role_name
```

Note

The commands used for ACL-based authorization do not support the [WITH GRANT OPTION] clause. If user A authorizes user B to access an object, user B cannot authorize user C to access the same object. Therefore, all authorization actions must be performed by one of the following types of users:

- Project owner
- Users who are assigned the admin role in a project
- Object creators in a project

Example of ACL-based authorization:

Scenario: Users with Alibaba Cloud accounts alice@aliyun.com and bob@aliyun.com are new members of the test_project project. In the test_project project, they must be able to submit jobs, create data tables, and view existing objects of the project. The administrator performs the following authorization actions:

```

use test_project
-- Open a project.
security
add user aliyun$alice@aliyun.com
-- Add alice@aliyun.com to the project.
add user aliyun$bob@aliyun.com
-- Add bob@aliyun.com to the project.
create role worker
-- Create a role.
grant worker TO aliyun$alice@aliyun.com
-- Assign the role to alice@aliyun.com.
grant worker TO aliyun$bob@aliyun.com
-- Assign the role to bob@aliyun.com.
grant CreateInstance, CreateResource, CreateFunction, CreateTable, List ON PROJECT test_project TO ROLE worker
-- Grant permissions to the role.

```

21.4.5. View permissions

MaxCompute allows you to view various types of permissions. The permissions include the permissions of users or roles and the access control list (ACL) of specified objects.

View the permissions of a user

Syntax:

```

show grants
-- View the access permissions of a user.
show grants for <username>
-- View the access permissions of a specified user. Only project owners and administrators can perform this operation.

```

View the permissions of a role

Syntax:

```

describe role <rolename>
-- View the access permissions that are granted to a specified role.

```

View the ACL of a specified object

Syntax:

```

show acl for <objectName> [on type <objectType>]
-- View the ACL of a specified object.

```

 **Note** If [on type <objectType>] is not specified, the object type is table.

Show permissions

MaxCompute uses the following characters to indicate the permissions of users or roles:

- **A**: Allow. Access is allowed.
- **D**: Deny. Access is denied.
- **C**: with Condition. This is conditional authorization. This character appears only for policy-based authorization.

For more information, see [Condition block structure](#).

- G: with Grant Option. Permissions on objects can be granted.

Example:

```
odps@test_project> show grants for aliyun$odpctest1@aliyun.com
[roles]
dev
Authorization Type: ACL
[role/dev]
A projects/test_project/tables/t1: Select [user/odpctest1@aliyun.com]
A projects/test_project: CreateTable | CreateInstance | CreateFunction | List
A projects/test_project/tables/t1: Describe | Select
Authorization Type: Policy
[role/dev]
AC projects/test_project/tables/test_*: Describe
DC projects/test_project/tables/alifinance_*: Select [user/odpctest1@aliyun.com]
A projects/test_project: Create* | List
AC projects/test_project/tables/alipay_*: Describe | Select
Authorization Type: ObjectCreator
AG projects/test_project/tables/t6: All
AG projects/test_project/tables/t7: All
```

21.5. Cross-project resource sharing

21.5.1. Overview

This topic describes resource sharing across projects.

You are the owner or administrator (with the admin role assigned) of a project, and a user applies for accessing resources of your project. If the applicant is a member of your project, we recommend that you use the user and authorization management features for your project. For more information, see [Project user and authorization management](#). Otherwise, you can share resources with the user across projects by using packages.

A package is used to share data and resources across projects. You can use a package to implement cross-project user authorization. The following example describes a scenario in which the package mechanism can be used.

Members of the Alifinance project need to access data of the Alipay project. The Alipay project administrator adds members of the Alifinance project to the Alipay project, and then grants the new members common permissions on the Alipay project. For security purposes, the Alipay project administrator does not want to authorize every user of the Alifinance project. A mechanism is required to allow the Alifinance project administrator to control access to the authorized objects.

If the package mechanism is used, the Alipay project administrator can package the objects that the Alifinance team needs to access, and then allow the package to be installed in the Alifinance project. After the package is installed, the Alifinance project administrator can determine whether to grant permissions on the package to members in the Alifinance project.

A package involves two subjects: package creator and package user. The package creator provides resources. The package creator packages the resources to be shared and the permissions to access these resources, and grants the package user the permissions to install and use the package. The package user consumes the resources. After the package user installs the package published by the package creator, the package user can directly access the resources.

The following topics describe the operations that can be performed by a package creator and package user.

21.5.2. Package usage

21.5.2.1. Operations for package creators

This topic describes the operations that package creators can perform.

Create a package

Run the following command to create a package:

```
create package <pkgname>
```

Delete a package

Run the following command to delete a package:

```
delete package <pkgname>
```

Add a resource that you want to share to a package

Run the following command to add a resource to a package:

```
add project_object to package package_name [with privileges <privileges>]
remove project_object from package package_name
project_object ::= table table_name | instance inst_name | function func_name | resource res_name
privileges ::= action_item1, action_item2, ...
```

Note

- The object type cannot be project. You cannot use a package of a project to create objects in other projects.
- In addition to the objects, the operation permissions on the objects are added to the package. If you do not specify permissions for an object by using [with privileges privileges], the object is read-only. You are granted only the READ, DESCRIBE, and SELECT permissions on the read-only objects. An object and its permissions are considered a whole. You can add or delete resources in a package. The permissions are revoked after resources are added or deleted.

Allow a project to use a package

Run the following command to allow a project to use a package:

```
allow project <prjname> to install package <pkgname> [using label <number>]
```

Revoke the permission for a project to use a package

Run the following command to revoke the permission for a project to use a package:

```
disallow project <prjname> to install package <pkgname>
```

View the packages that are already created and installed

Run the following command to view the packages that are already created and installed:

```
show packages
```

View details of a package

Run the following command to view details of a package:

```
describe package <pkgname>
```

21.5.2.2. Operations of package users

This topic describes the operations that are performed by package users.

The installed package is a type of independent object in MaxCompute. Resources in a package are those of other projects that are shared with you. To access resources in a package, you must have the permission to read the package. If you do not have this permission, request the project owner or user admin to grant the permission to you. The project owner or user admin can grant the permission by using ACL- or policy-based authorization.

Example of ACL-based authorization: If you are the project owner or user admin, you can run the following commands to allow a user with the Alibaba Cloud account `odps_test@aliyun.com` to access resources in a package.

```
use prj2 security
install package prj1.testpkg
grant read on package prj1.testpackage to user aliyun$odps_test@aliyun.com
```

Example of policy-based authorization: If you are the project owner or user admin, you can run the following commands to allow all users in the `prj2` project to access resources in the package.

```
use prj2
install package prj1.testpkg
put policy /tmp/policy.txt
```

The `policy.txt` file in the `/tmp` directory contains the following content:

```
{
  "Version": "1", "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "odps:Read", "Resource": "acs:odps:*:projects/prj2/packages/prj1.testpkg"
  }]
}
```

Install a package

Run the following command to install a package:

```
install package <pkgname>;
```

 **Note** In the preceding command, `pkgname` must be in the format of `<projectName>.<packageName>`.

Uninstall a package

Run the following command to uninstall a package:

```
uninstall package <pkgname>;
```

 **Note** In the preceding command, `pkgname` must be in the format of `<projectName>.<packageName>`.

View packages

Run the following commands to view packages:

```
show packages
-- View the packages that you have already created and installed.
describe package <pkgname>
-- View details of a package.
```

21.6. Project protection

21.6.1. Overview

This topic describes the project data protection mechanism.

Some enterprises, such as financial institutions and military enterprises, have high data security requirements. For example, their employees can perform their jobs only in the workplace, and are not allowed to take work materials out of the office. All USB ports on office computers are disabled. These measures aim to prevent leaks of sensitive data.

For example, you are a MaxCompute project administrator in charge of a project with sensitive data. The data must not be shared to other projects. In this case, you are required to perform configurations and operations for project data protection.

21.6.2. Project data protection

This topic describes the project data protection mechanism of MaxCompute.

MaxCompute provides a project data protection mechanism. This mechanism can forbid all operations that may cause data transfer to other projects. You need only to run the following command to enable the data protection mechanism for your project.

```
set security.ProjectProtection=true;
-- Set security.ProjectProtection to true. This ensures that data can be used within a project and cannot be transferred to other projects. This setting does not take effect on data in external tables .
-- You can also run the following commands to enable the project data protection mechanism.
set security.ProjectProtection=true;
set project IDENTIFY_EXTERNAL_TABLE_WRITE_AS_DATALEAK=true;
-- After you set security.ProjectProtection to true, set project IDENTIFY_EXTERNAL_TABLE_WRITE_AS_DATALEAK to true. This means that project data cannot be written to external storage sources by using external tables .
```

After the project data protection mechanism is enabled, data of your project cannot be transferred to other projects.

By default, `security.ProjectProtection` is set to `false`, which indicates that the project data protection mechanism is disabled. Users who are authorized to access multiple projects can perform cross-project data access operations to transfer data. If a project stores highly-sensitive data, the project administrator must configure the project data protection mechanism.

21.6.3. Data export methods after project data protection is enabled

This topic describes the data export methods that you can use after project data protection is enabled for MaxCompute.

After data protection is enabled for your project, you may soon encounter this situation: Alice submits a request to export the data of a table from the project. You approve the request from Alice because this table does not contain sensitive data. To meet the business requirements of Alice, MaxCompute provides two methods to export data after project data protection is enabled.

Configure an exception policy

If a project owner enables project data protection, the owner can run the following command to configure an exception policy:

```
set ProjectProtection=true with exception <policyFile>
```

Note The exception policy is different from the policy configured for policy-based authorization even though the two policies have the same syntax. This exception policy acts as an exception to the rules of project data protection. After the exception policy is configured, the access requests that meet the description of the exception policy are ignored by the project data protection rules.

Example of an exception policy:

The following policy allows Alice with the Alibaba Cloud account Alice@aliyun.com to export data to the alipay project when Alice uses SQL tasks to perform SELECT operations on the alipay.table_test table.

```
{
  "Version": "1", "Statement": [{
    "Effect": "Allow", "Principal": "ALIYUN$Alice@aliyun.com",
    "Action": ["odps:Select"],
    "Resource": "acs:odps:*:projects/alipay/tables/table_test",
    "Condition": {
      "StringEquals": { "odps:TaskType": ["DT", "SQL"]
    }
  }
}]
}
```

 Note

- The exception policy is not a common authorization method. If Alice does not have the SELECT permission on the alipay.table_test table, Alice cannot export data even if the preceding exception policy is configured. Project data protection is a method to control data transfer but does not control data access. Project data protection is effective only if users can access their destination data.
- After an exception policy is configured, data leaks may occur due to time-of-check to time-of-use (TOCTOU). TOCTOU is also considered a race condition. Symptom:
 - i. [TOC stage] User A submits an application to the project owner to export the t1 table. After the owner verifies that the t1 table does not contain sensitive data, the owner configures an exception policy to authorize user A to export the t1 table.
 - ii. A malicious user writes sensitive data to the t1 table.
 - iii. [TOU stage] User A exports the t1 table. The t1 table exported by user A is not the table verified by the project owner.

To prevent this issue, the project owner must make sure that all project members, including the users who are assigned the admin role, do not have the UPDATE permission or the DROP and CREATE TABLE permissions on the table that a user wants to export. In the preceding example, we recommend that the project owner create a snapshot of the t1 table in Step i, and then use this snapshot to configure an exception policy. In addition, the project owner does not assign the admin role to other project members.

Configure a trusted project

If data protection is enabled for a project, data of the project can be exported to its trusted project. This export does not violate the rules of project data protection. If multiple projects are configured as trusted projects for each other, they form a trusted project group. Data of a project can be exported to other projects in the group but cannot be exported to the projects out of the group.

You can run the following commands to manage the trusted projects:

```
list trustedprojects
-- List all trusted projects that are configured for a project.
add trustedproject <projectname>
-- Add a project as a trusted project for the project.
remove trustedproject <projectname>
-- Remove a trusted project from the project.
```

21.6.4. Package-based resource sharing and project data protection

This topic describes the relationship between package-based resource sharing and data protection in MaxCompute projects.

In MaxCompute, package-based resource sharing and project data protection are mutually independent mechanisms that take effect at the same time, but their features are mutually exclusive.

In MaxCompute, package-based resource sharing takes precedence over project data protection. If a data object in a project is shared with users in other projects based on resource sharing, the project data protection rules will not apply to this object.

To prevent data transfer from a project to another, you must check the following items after you set ProjectProtection to True:

- Make sure that no trusted projects are added. If trusted projects are added, evaluate possible risks.
- Make sure that no exception policies are configured. If exception policies are configured, evaluate possible risks, especially the risks caused by time-of-check to time-of-use (TOCTOU).
- Check whether package-based data sharing is not in use. If package-based data sharing is in use, make sure that the package contains no sensitive data.

21.7. Project security configurations

This topic describes security configurations for MaxCompute projects.

MaxCompute is a platform that allows multiple tenants to process data at the same time. When these tenants process data on MaxCompute, they may have different requirements for data security. To meet their requirements, MaxCompute allows tenants to configure security settings for projects. Project owners can specify an external account system that is supported by MaxCompute and an authentication model to suit their needs.

MaxCompute supports multiple orthogonal authorization mechanisms, such as ACL-based authorization, policy-based authorization, and implicit authorization. If implicit authorization is used, an object creator is automatically granted the permissions to access the object. However, not all users require these security mechanisms. You can configure an authentication model based on your business requirements and usage habits.

```
show SecurityConfiguration
-- View the security configuration of a project.
set security.CheckPermissionUsingACL=true/false
-- Enable or disable ACL-based authorization. The default value of security.CheckPermissionUsingACL
is true.
set security.CheckPermissionUsingPolicy=true/false
-- Enable or disable policy-based authorization. The default value of security.CheckPermissionUsingP
olicy is true.
set security.ObjectCreatorHasAccessPermission=true/false
-- Allow or disallow an object creator to be granted the object access permission by default. The de
fault value of security.ObjectCreatorHasAccessPermission is true.
set security.ObjectCreatorHasGrantPermission=true/false
-- Allow or disallow an object creator to be granted the authorization permission by default. The de
fault value of security.ObjectCreatorHasGrantPermission is true.
set security.LabelSecurity=true/false
-- Enable or disable the label security policy.
set security.ProjectProtection=true/false
-- Enable or disable project data protection to allow or disallow data transfer to other projects.
```

21.8. Authorization policies

21.8.1. Policy overview

This topic describes the policies that are used for policy-based authorization.

Policy-based authorization is based on principal. A principal can be a user or role. Policies are considered resources of a principal. You can perform policy-based authorization only on a principal that exists. If a principal is deleted, the policies of the principal are automatically deleted. Policy-based authorization uses a custom policy language defined by MaxCompute to perform authorization operations on principals. After the authorization is complete, principals are allowed or not allowed to access resources in projects.

Policy-based authorization is a new authorization mechanism. It is used to handle complicated scenarios in which ACL-based authorization struggles to deal with, such as:

- A principal is granted permissions to access a group of resources, such as all functions and all tables whose name starts with taobao, at a time.
- A principal is granted permissions with specified conditions, for example, the permission on a table is effective

at a specified time, for a request from a specified IP address, or for only SQL tasks (not supported for other types of tasks)

Command syntax:

```
GET POLICY;
-- Read the policies of a project.
PUT POLICY <policyFile>;
-- Set a policy for the project and use this policy to overwrite an existing policy.
GET POLICY ON ROLE <roleName>;
-- Read the policies of a role in the project.
PUT POLICY <policyFile> ON ROLE <roleName>;
-- Set a policy for a role in the project and use this policy to overwrite an existing policy.
```

Note MaxCompute supports project policies and role policies. A project policy applies to all users of the project, whereas a role policy applies only to users who are assigned roles. You must specify a user principal for project policies. You cannot specify a user principal for role policies because the principal has been assigned a role.

Example of configuring a project policy for policy-based authorization:

Scenario: A project policy needs to be configured to ensure that the user with the Alibaba Cloud account `alice@aliyun.com` can only submit a request from the CIDR block `10.32.180.0/23` before `2019-11-11 23:59:59`. The user can perform only the operations to create instances and tables and view the operations that are performed on the `test_project` project. The user is not allowed to delete tables from the `test_project` project.

The following project policy is configured:

```
{
  "Version": "1", "Statement": [{
    "Effect": "Allow", "Principal": "ALIYUN$alice@aliyun.com",
    "Action": ["odps:CreateTable", "odps:CreateInstance", "odps:List"],
    "Resource": "acs:odps:*:projects/test_project",
    "Condition": { "DateLessThan": {
      "acs:CurrentTime": "2017-11-11T23:59:59Z"
    }
    },
    "IpAddress": { "acs:SourceIp": "10.32.180.0/23"
    }
  }
  ],
  {
    "Effect": "Deny", "Principal": "ALIYUN$alice@aliyun.com", "Action": "odps:Drop", "Resource": "acs:odps:*:projects/test_project/tables/*"
  }
]
}```json
```

```

```json
{
 "Version": "1", "Statement": [{
 "Effect": "Allow", "Principal": "ALIYUN$alice@aliyun.com",
 "Action": ["odps:CreateTable", "odps:CreateInstance", "odps:List"],
 "Resource": "acs:odps:*:projects/test_project",
 "Condition": { "DateLessThan": {
 "acs:CurrentTime": "2017-11-11T23:59:59Z"
 }
 },
 "IpAddress": { "acs:SourceIp": "10.32.180.0/23"
 }
 }
],
 {
 "Effect": "Deny", "Principal": "ALIYUN$alice@aliyun.com",
 "Action": "odps:Drop",
 "Resource": "acs:odps:*:projects/test_project/tables/*"
 }
]
}

```

```

{
 "Version": "1", "Statement": [{
 "Effect": "Allow", "Principal": "ALIYUN$alice@aliyun.com",
 "Action": ["odps:CreateTable", "odps:CreateInstance", "odps:List"],
 "Resource": "acs:odps:*:projects/test_project",
 "Condition": { "DateLessThan": {
 "acs:CurrentTime": "2017-11-11T23:59:59Z"
 }
 },
 "IpAddress": { "acs:SourceIp": "10.32.180.0/23"
 }
 }
],
 {
 "Effect": "Deny", "Principal": "ALIYUN$alice@aliyun.com",
 "Action": "odps:Drop",
 "Resource": "acs:odps:*:projects/test_project/tables/*"
 }
]
}

```

#### Note

- Only role policies and project policies are supported. User policies are not supported.
- Only one policy file can be configured for each type of policy. When the PUT POLICY command is executed to configure a policy, the configured policy overwrites an existing policy. To modify a policy, perform the following steps:
  - i. Run the GET POLICY command.
  - ii. Manually merge policy statements.
  - iii. Run the PUT POLICY command.

## 21.8.2. Policy-related terminology

This topic describes the basic terms used in a policy.

Permission is a basic concept of access control. If a requester wants to take an action on a resource, the action may be allowed or denied, based on the permission settings. A statement refers to the formal description of a single permission. A policy refers to a set of statements.

An access policy consists of the following access control elements: principal, action, resource, access restriction, and effect. The following sections provide brief descriptions of these elements.

## Principal

A principal of an object is a user or role to which permissions are assigned in an access policy. For example, the access policy allows Michael to perform the CreateObject action on the resource SampleBucket before December 31, 2019. Michael is the principal of the object.

## Action

An action is an activity that the principal has permissions to perform. For example, the access policy allows Michael to perform the CreateObject action on the resource SampleBucket before December 31, 2019. CreateObject is an action of the access policy.

## Resource

A resource is the object that a principal requests access to. For example, the access policy allows Michael to perform the CreateObject action on the resource SampleBucket before December 31, 2019. SampleBucket is a resource of the access policy.

## Access restriction

Access restriction is the condition that specifies whether a permission takes effect. For example, the access policy allows Michael to perform the CreateObject action on resource SampleBucket before December 31, 2019. The access restriction is before December 31, 2019.

## Effect

Authorization effect has two options: allow or deny. Deny takes precedence over allow during permission checks.

 **Notice** Deny and revocation of authorization are different. Revocation of authorization involves revocation of allow and deny. For example, conventional databases support the Revoke and Revoke Deny actions.

## 21.8.3. Access policy structure

### 21.8.3.1. Overview

This topic describes the language structure of policies defined for policy-based authorization.

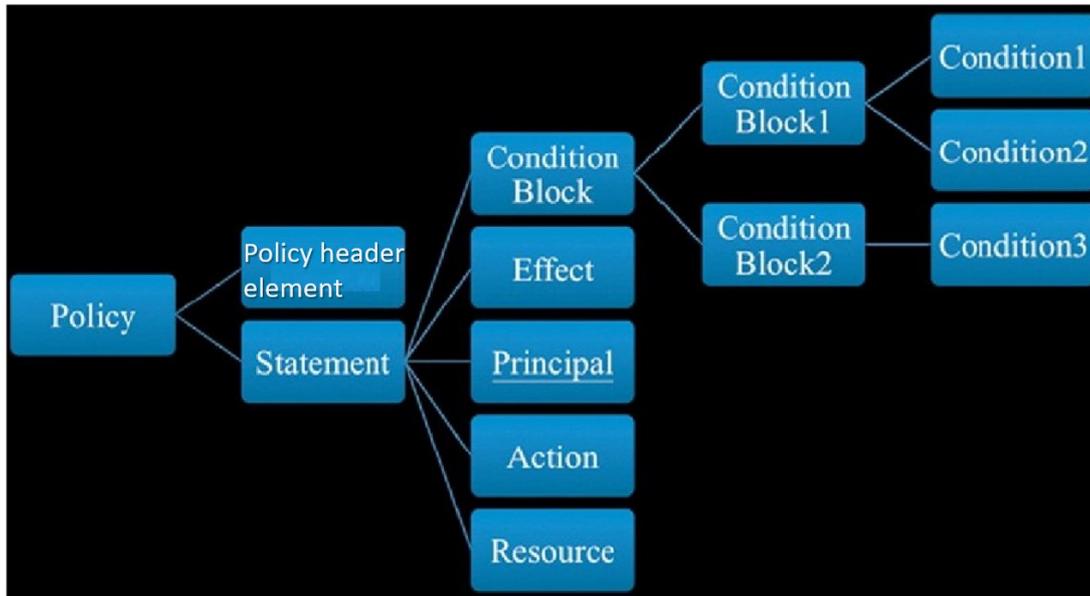
A policy consists of the following parts:

- An optional policy header
- One or more statements

 **Note** The policy header is optional and includes the policy version. The policy body is a set of statements.

The following figure shows the structure of a policy.

Structure of a policy



### 21.8.3.2. Authorization statement structure

This section describes the structure of the authorization statement in policies defined for policy-based authorization.

An authorization statement includes the following entries:

- **Effect**: indicates the permission type of this statement. The value can be Allow or Deny.
- **Principal**: If a policy is bound to a user or role in the authorization process, such as the role policy of MaxCompute, you cannot specify a principal. If a policy is bound to a project or objects of the project in the authorization process, such as the project policy of MaxCompute, you must specify a principal.
- **Action**: indicates the authorization operation. It can be one or multiple operation names and supports the asterisk (\*) and question mark (?). For example, `Action = *` indicates all operations.
- **Resource**: indicates the authorization object. It can be one or multiple object names and supports the asterisk (\*) and question mark (?). For example, `Resource = *` indicates all objects.
- **Condition block**: indicates the conditions that must be met for the permission described by this authorization statement to take effect.

### 21.8.3.3. Structure of condition blocks

This topic describes the structure of condition blocks in the statements that are used for policy-based authorization.

A condition block consists of one or more condition clauses. A condition clause consists of a condition action type, condition key, and condition value. The condition action types and condition keys are described in detail in subsequent topics.

The following rules are used to check whether conditions in a condition block are met:

- A condition key can correspond to one or more values. If the value of the condition key is equal to one of the specified values, the condition is met.
- If multiple condition keys in a clause of a condition action type are met, this clause is met.
- If condition keys in all condition clauses of a condition block are met, this condition block is met.

### 21.8.3.4. Condition action types

This topic describes the types of condition actions in policy condition clauses.

The actions on the following data types are supported: STRING, BOOLEAN, and IP address as well as the numeric, date, and time types. Methods supported by each action type:

### STRING

```
StringEquals
StringNotEquals
StringEqualsIgnoreCase
StringNotEqualsIgnoreCase
StringLike
StringNotLike
```

### Numeric

```
NumericEquals
NumericEquals
NumericLessThan
NumericLessThanEquals
NumericGreaterThan
NumericGreaterThanEquals
```

### Date and Time

```
DateEquals
DateNotEquals
DateLessThan
DateLessThanEquals
DateGreaterThan
DateGreaterThanEquals
```

### BOOLEAN

```
Bool
```

### IP Address

```
IpAddress NotIpAddress
```

## 21.8.3.5. Condition keys

This topic describes the condition keys in policy condition clauses.

MaxCompute supports the condition keys reserved by Alibaba Cloud Service (ACS). The following table describes these condition keywords.

### Condition keys

Condition key reserved by ACS	Data type	Description
<code>acs:CurrentTime</code>	Date and Time	The time when the web server receives a request. The time is based on the ISO 8601 standard, for example, 2019-11-11T23:59:59Z.

Condition key reserved by ACS	Data type	Description
<b>acs:SecureTransport</b>	BOOLEAN	Whether the request is sent over a secure channel, such as an HTTPS channel.
<b>acs:SourceIp</b>	IP Address	The IP address of the client that sends a request.
<b>acs:UserAgent</b>	STRING	The User-Agent header in a request.
<b>acs:Referer</b>	STRING	The HTTP referer in a request.

 **Note** `acs:SourceIp` indicates the remote IP address of an HTTP connection. This condition key does not indicate an IP address in the IP address list represented by `x-forwarded-for` in the HTTP header. For example, if `10.230.205.105` is a LAN IP address, `acs:SourceIp` indicates the egress gateway IP address of this LAN. If the network egress uses a proxy server, `acs:SourceIp` indicates the IP address of the proxy server. If the request traverses across multiple proxy servers, `acs:SourceIp` indicates the IP address of the final proxy server. The value of `acs:SourceIp` may vary based on the rules configured on proxy servers.

## 21.8.4. Access policy norm

### 21.8.4.1. Principal naming conventions

This topic describes the principal naming conventions in language specifications of policies defined for policy-based authorization.

The principal is the request sender. Only an Alibaba Cloud account, domain account, or Taobao account can be used as a principal. An Alibaba Cloud account can be represented by ID or `DisplayName`.

Example:

```
"Principal": "43274"
"Principal": "ALIYUN$bob@aliyun.com"
"Principal": ["ALIYUN$bob@aliyun.com", "ALIYUN$jack@aliyun.com", "TAOBAO$alice"]
```

### 21.8.4.2. Naming conventions of resources

This topic describes the naming conventions of resources in language specifications of policies defined for policy-based authorization.

MaxCompute resources are named in the following format:

```
acs:<service-name>:<namespace>:<relative-id>
```

The following table describes the parameters in this format.

#### Parameters

Parameter	Description
<b>acs</b>	The retained resource header.
<b>service-name</b>	The name of an Alibaba Cloud service (ACS), such as MaxCompute, Object Storage Service (OSS), and Tablestore.

Parameter	Description
<b>namespace</b>	The namespace that is used for resource isolation. If an Alibaba Cloud account is used for resource isolation, the value can be an Alibaba Cloud account. If this parameter is not required, you can use an asterisk (*) instead.
<b>relative-id</b>	The service-related resource. Its meaning is based on services. The value format of this parameter supports a tree structure that is similar to the file path. The following value format of relative-id is used for MaxCompute: <pre>projects/&lt;project_name&gt;/&lt;object_type&gt;/&lt;object_name&gt;</pre>

The following table provides examples of MaxCompute resource names.

### Examples of MaxCompute resource names

Name	Description
<b>*</b>	Indicates all objects in the project.
<b>projects/prj1/tables/t1</b>	Indicates the t1 table of the prj1 project.
<b>projects/prj1/instances/*</b>	Indicates all instances in the prj1 project.
<b>projects/prj1/tables/*</b>	Indicates all tables in the prj1 project.
<b>projects/prj1/tables/taobao*</b>	Indicates all tables in the prj1 project. The table names are all prefixed by taobao.

### 21.8.4.3. Naming conventions of actions

This topic describes the naming conventions of actions in language specifications of policies defined for policy-based authorization.

Naming convention of actions:

```
<service-name>:<action-name>
```

Parameters:

- **service-name**: the name of an Alibaba Cloud service, for example, MaxCompute, Object Storage Service (OSS), and Tablestore.
- **action-name**: the name of the service-related action.

The following table provides examples of MaxCompute action names.

### Examples of MaxCompute action names

Action name	Description
<b>*</b>	Indicates all actions.
<b>odps:*</b>	Indicates all MaxCompute actions.
<b>odps:CreateTable</b>	Indicates the CreateTable action of MaxCompute.
<b>odps:Create*</b>	Indicates all MaxCompute actions whose names start with Create.

## 21.8.4.4. Naming conventions of condition keys

This topic describes the naming conventions of condition keys in language specifications of policies defined for policy-based authorization.

Condition keys that are reserved for Alibaba Cloud Services (ACSs) are named in the following format:

```
acs:<condition-key>
```

Parameter: condition-key: the condition key, which is accessible to all ACSs. The following condition keys are supported: acs:CurrentTime, acs:SecureTransport, acs:SourceIp, acs:UserAgent, and acs:Referer.

Condition keys that are specific to an ACS are named in the following format:

```
<service-name>:<condition-key>
```

Parameter: condition-key: the condition key that is specific to an ACS.

## 21.8.4.5. Example of a policy for policy-based authorization

This topic provides an example of a policy for policy-based authorization.

Sample policy:

```
{
 "Version": "1",
 "Statement": [{
 "Effect": "Allow",
 "Principal": "ALIYUN$alice@aliyun.com",
 "Action": ["odps:CreateTable", "odps:CreateInstance", "odps:List"],
 "Resource": "acs:odps:*:projects/prj1",
 "Condition": { "DateLessThan": {
 "acs:CurrentTime": "2019-11-11T23:59:59Z"
 } },
 "IpAddress": { "acs:SourceIp": "10.32.180.0/23"
 }
 }],
 {
 "Effect": "Deny", "Principal": "ALIYUN$alice@aliyun.com",
 "Action": "odps:Drop",
 "Resource": "acs:odps:*:projects/prj1/tables/*"
 }
]
```

**Note** The user with the Alibaba Cloud account `alice@aliyun.com` is authorized to submit a request from CIDR block `10.32.180.0/23` only before `2019-11-11T23:59:59Z`. The user has only the `CREATE INSTANCE`, `CREATE TABLE`, and `LIST` permissions on the `prj1` project. The user is not allowed to delete tables from the `prj1` project.

## 21.8.5. Differences between policy-based access control and ACL-based access control

This topic describes the differences between policy-based access control and ACL-based access control.

## ACL-based access control

- Before you grant or revoke permissions, make sure that the grantee, such as a user or role and an object such as a table exist. This also applies to authorization in Oracle and the purpose is to avoid the security risks that may occur after you delete and recreate an object with the same name.
- If you delete an object, all permissions related to the object are automatically revoked.
- Only allow (allowlist) authorization is supported. Deny (denylist) authorization is not supported.
- The GRANT and REVOKE commands are used for ACL-based access control. These commands are easy to use and not prone to mistakes. Conditional authorization is not supported.
- ACL-based access control is suitable for simple scenarios where conditional authorization or a deny action is not required, and only the existing objects to which the permissions are granted.

## Policy-based access control

- Before you grant or revoke permissions, you are not required to check whether the grantee or object exists. You can use an asterisk (\*) to represent an object. For example, *projects/tbproj/tables/taobao\** indicates all tables whose names start with taobao in the tbproj project. Similar to authorization in MySQL, policy-based access control allows you to grant permissions to a non-existent object. The grantee must consider the security risks that may occur after you delete and recreate an object with the same name.
- If you delete an object, all permissions related to the object are not revoked.
- Both allow (allowlist) authorization and deny (denylist) authorization are supported. If allow (allowlist) authorization and deny (denylist) authorization are performed at the same time, the deny (denylist) authorization takes precedence.
- Conditional authorization is supported. The grantee can enforce a maximum of 20 conditions on allow or deny authorization. For example, these conditions can be used to specify a valid range of IP addresses for requestors and the access time that must be earlier than the specified time.
- Policy-based access control is suitable for relatively complicated scenarios where conditional authorization and deny actions are required for non-existent objects.
- The commands for policy-based access control are used and these commands are complex.

## 21.8.6. Limits

This topic describes the limits of policy-based authorization.

### Limits

Item	Upper Limit	Description
ACCESS_POLICY_SIZE_LIMIT	32 KB	The maximum size of the text of a policy.
USER_NUMBER_LIMIT_IN_ON E_PROJECT	1000	The maximum number of users that can be added to a project.
ROLE_NUMBER_LIMIT_IN_ON E_PROJECT	500	The maximum number of roles that can be created for a project.
ROLE_NAME_LENGTH_LIMIT	64 KB	The maximum character length of a role name.
SECURITY_COMMENT_SIZE_L IMIT	1 KB	The maximum character length of a comment.
PACKAGE_NAME_LENGTH_LI MIT	128 KB	The maximum character length of a package name.

Item	Upper Limit	Description
ALLOW_PROJECT_NUMBER_LIMIT_IN_ONE_PACKAGE	1024	The maximum number of projects for which a package can be installed.
RESOURCE_NUMBER_LIMIT_IN_ONE_PACKAGE	256	The maximum number of resources that can be included in a package.
PACKAGE_NUMBER_LIMIT_IN_ONE_PROJECT	512	The maximum number of packages that can be created for a project.
INSTALLED_PACKAGE_NUMBER_LIMIT_IN_ONE_PROJECT	64	The maximum number of packages that can be installed for a project.

## 21.9. Collection of security statements

### 21.9.1. Project security configurations

This topic describes the statements that are used for security configurations of projects.

#### Authentication configurations

##### Statements for authentication configurations

Statement	Description
show SecurityConfiguration	Allows you to view security configurations of a project.
set CheckPermissionUsingACL=true/false	Allows you to enable or disable ACL-based authorization.
set CheckPermissionUsingPolicy=true/false	Allows you to enable or disable policy-based authorization.
set ObjectCreatorHasAccessPermission=true/false	Allows you to grant or revoke default access permissions to or from object creators.
set ObjectCreatorHasGrantPermission=true/false	Allows or disallows an object creator to be granted the ACL-based authorization permission by default.

#### Project data protection

##### Statements for project data protection

Statement	Description
set ProjectProtection=false	Allows you to disable project data protection.
set ProjectProtection=true [with exception <policy>]	Allows you to enable project data protection.
list TrustedProjects	Allows you to view trusted projects.
add TrustedProject <projectName>	Allows you to add a project to trusted projects.
remove TrustedProject <projectName>	Allows you to remove a project from trusted projects.

## 21.9.2. Project permission management

This topic describes the statements used for permission management of projects.

### User management

#### Statements

Statement	Description
<code>list users</code>	Allows you to view all users that are added to a project.
<code>add user &lt;username&gt;</code>	Allows you to add a user.
<code>remove user &lt;username&gt;</code>	Allows you to remove a user.

### Role management

#### Statements

Statement	Description
<code>list roles</code>	Allows you to view all the existing roles.
<code>create role &lt;rolename&gt;</code>	Allows you to create a role.
<code>drop role &lt;rolename&gt;</code>	Allows you to delete a role.
<code>grant &lt;rolelist&gt; to &lt;username&gt;</code>	Allows you to revoke roles from a user.
<code>revoke &lt;rolelist&gt; from &lt;username&gt;</code>	Allows you to assign one or multiple roles to a user.

### ACL-based authorization

#### Statements

Statement	Description
<code>grant &lt;privList&gt; on &lt;objType&gt; &lt;objName&gt; to user &lt;username&gt;</code>	Allows you to grant permissions to a user
<code>grant &lt;privList&gt; on &lt;objType&gt; &lt;objName&gt; to role &lt;rolename&gt;</code>	Allows you to grant permissions to a role.
<code>revoke &lt;privList&gt; on &lt;objType&gt; &lt;objName&gt; from user &lt;username&gt;</code>	Allows you to revoke permissions from a user.
<code>revoke &lt;privList&gt; on &lt;objType&gt; &lt;objName&gt; from role &lt;rolename&gt;</code>	Allows you to revoke permissions from a role.

### Policy-based authorization

#### Statements

Statement	Description
<code>get policy</code>	Allows you to view policy settings of a project.
<code>put policy &lt;policyFile&gt;</code>	Allows you to configure a policy for a project
<code>get policy on role &lt;roleName&gt;</code>	Allows you to view the policy settings of a role.
<code>put policy &lt;policyFile&gt; on role &lt;roleName&gt;</code>	Allows you to configure a policy for a role.

## Permission review

### Statements

Statement	Description
<code>whoami</code>	Allows you to view the information about a user.
<code>show grants [for &lt;username&gt;] [on type &lt;objectType&gt;]</code>	Allows you to view the permissions and roles of a user.
<code>show acl for &lt;objectName&gt; [on type &lt;objectType&gt;]</code>	Allows you to view the authorization information of an object.
<code>describe role &lt;roleName&gt;</code>	Allows you to view the authorization and assignment information of a role.

## 21.9.3. Package-based resource sharing

This topic describes the statements that are used for package-based resource sharing.

### Share resources

#### Statements for sharing resources

Statement	Description
<code>create package &lt;pkgName&gt;</code>	Allows you to create a package.
<code>delete package &lt;pkgName&gt;</code>	Allows you to delete a package.
<code>add &lt;objType&gt; &lt;objName&gt; to package&lt;pkgName&gt; [with privileges privs]</code>	Allows you to add resources that you want to share to a package.
<code>remove &lt;objType&gt; &lt;objName&gt; from package &lt;pkgName&gt;</code>	Allows you to remove shared resources from a package.
<code>allow project &lt;prjName&gt; to install package &lt;pkgName&gt; [using label &lt;num&gt;]</code>	Allows a project to use a package of a specified user.
<code>disallow project &lt;prjName&gt; to install package &lt;pkgName&gt;</code>	Disallows a project from using the package.

### Use resources

#### Statements for using resources

---

Statement	Description
<code>install package &lt;pkgName&gt;</code>	Allows you to install a package.
<code>uninstall package &lt;pkgName&gt;</code>	Allows you to uninstall a package.

## View packages

### Statements for viewing packages

Statement	Description
<code>show packages</code>	Allows you to view all the packages that are created and installed.
<code>describe package &lt;pkgName&gt;</code>	Allows you to view the details of a package.

# 22. Hierarchical throttling

## 22.1. Overview

MaxCompute provides user-based job throttling.

If some users perform inappropriate operations, a series of issues may arise. For example, if a user submits a large number of jobs, the jobs may be queued for a long period of time and resources in the resource group may be exhausted. To address these issues, MaxCompute provides user-based job throttling in this version and later. After user-based job throttling is enabled, the project owner can specify an upper limit for the number of jobs that can be concurrently submitted based on the resources required for the business or teams.

## 22.2. Hierarchical throttling

MaxCompute supports hierarchical throttling.

MaxCompute uses global throttling for system-wide configurations. To properly control project resources, MaxCompute also allows project owners to use hierarchical throttling for a project.

 **Note** Project owners can use hierarchical throttling rules to specify the maximum number of instances that are allowed for a project.

## 22.3. Specify the maximum number of instances that are allowed for a project

This topic describes how to use hierarchical throttling rules to specify the maximum number of instances that are allowed for a project.

### Formulate JSON rules for throttling

Assume that the following throttling rules are defined:

- test\_rule1: indicates that the maximum number of instances that users except user1 can run in the information\_schema\_dev project at the same time is two.
- test\_rule2: indicates that the maximum number of instances that user1 can run in the information\_schema\_dev project at the same time is four.

If the user field is set to a specified user other than an asterisk (\*), the priority of the specified user is higher than the priorities of other users. After the two rules take effect, users except user1 can run a maximum of two instances in the information\_schema\_dev project at the same time. However, user1 can run a maximum of four instances in the information\_schema\_dev project at the same time.

In addition, JSON rules of a project must be written in the same list.

Sample rules:

```
[{
 "name": "test_rule1",
 "entity": {
 "project": "information_schema_dev",
 "user": "*"
 },
 "settings": [{
 "metric": "total_instances",
 "threshold": "2",
 "action": "deny"
 }]
},
{
 "name": "test_rule2",
 "entity": {
 "project": "information_schema_dev",
 "user": "user1"
 },
 "settings": [{
 "metric": "total_instances",
 "threshold": "4",
 "action": "deny"
 }]
}]
```

 **Note** In the sample rules, you must specify name, project, user, and threshold. Other fields are reserved for future use and must retain their default values.

## Configure and view throttling rules

Convert the preceding sample rules in the JSON format into a single row, and run the `setproject` command in the MaxCompute console to configure throttling rules.

```
information_schema_dev> setproject THROTTLING_RULES=[{"name":"test_rule1","entity":{"project":"information_schema_dev","user":"*"},"settings":[{"metric":"total_instances","threshold":"2","action":"deny"}]},{"name":"test_rule2","entity":{"project":"information_schema_dev","user":"19655xxxxxx48481"},"settings":[{"metric":"total_instances","threshold":"4","action":"deny"}]}];
```

Run the following command to view the throttling rules:

```
information_schema_dev> setproject;
```

## View the output information for throttling

If a large number of instances that meet the throttling rules run in the `information_schema_dev` project, an error is returned. Example of error information:

```
FAILED: Request rejected by flow control. Break rule: test_rule2. Matched instance count: 10. Limit: 4
-- The request is rejected because test_rule 2 is violated. The number of instances that match this rule is 10. However, this rule specifies that a maximum of 4 instances can run at the same time.
```

## Delete throttling rules

You can leave `THROTTLING_RULES` empty to delete the throttling rules.

```
information_schema_dev> setproject THROTTLING_RULES=[];
```

## Usage notes

For ease of description, rules can be written in the following format: `/project/user/empid? limit=x`. Wildcards are allowed in rules. If a rule with wildcards matches a throttling rule, a rule priority issue may occur.

In this case, the throttling rule has a higher priority than the rule with wildcards. The rule priority issue occurs only if a rule with wildcards matches a throttling rule. In other cases, the rule with wildcards and the throttling rule take effect at the same time.

Example 1: A throttling rule takes effect.

- Rule 1: `/meta_dev/*? limit=10`
- Rule 2: `/meta_dev/123456789? limit=100`

In this example, Rule 1 matches Rule 2. Rule 2 is a throttling rule and takes effect. Rule 1 is a rule with wildcards and does not take effect. The user with the ID of 123456789 in the meta\_dev project can run a maximum of 100 instances at the same time. Other users can run a maximum of 10 instances at the same time.

Example 2: Two throttling rules take effect at the same time.

- Rule 3: `/meta_dev? limit=10`
- Rule 4: `/meta_dev/123456789? limit=100`

In this example, the two rules take effect at the same time. The two rules do not contain wildcards. Therefore, no rule priority issues occur. The user with the ID of 123456789 in the meta\_dev project can run a maximum of 10 instances at the same time. This is because Rule 3 indicates that the total number of instances that all users in the meta\_dev project can run at the same time is 10. This rule applies even if the user with the ID of 123456789 is allowed to run 100 instances at the same time.

Example 3: A throttling rule and a rule with wildcards take effect at the same time.

- Rule 5: `/meta_dev/*? limit=10`
- Rule 6: `/meta_dev/123456789/111111? limit=100`

In this example, the two rules take effect at the same time. Rule 5 is a rule with wildcards but does not match Rule 6. This is because a rule with wildcards matches only a throttling rule at the same directory level. Therefore, no rule priority issues occur.

# 23. Frequently-used tools

## 23.1. MaxCompute console

### 23.1.1. Usage notes

This topic provides usage notes for the MaxCompute client.

- Do not parse data based on the output format of the MaxCompute client. The client output format may not be forward compatible. The command syntax and behavior vary based on client versions.
- The MaxCompute client is a Java program. It requires JRE to run. You must download and install JRE 1.8 to run the MaxCompute client.
- Before you configure the client, make sure that a project is created by using an Alibaba Cloud account and the AccessKey ID and AccessKey secret of the account are obtained.
- For more information about how to use the MaxCompute client, see [Configure the client](#).

### 23.1.2. Install the MaxCompute client

This topic describes how to install the MaxCompute client.

1. Download the [client](#) package to your computer.
2. Decompress the package to the folder where you want to store the client. The package contains the following folders:

```
bin/
conf/
lib/
plugins/
```

3. Edit the following information in the `odps_config.ini` file in the `conf` folder:

```
project_name=my_project
access_id=*****
access_key=*****
end_point= <Endpoint of MaxCompute>
```

- Set `access_id` to the AccessKey ID of your Alibaba Cloud account and `access_key` to the AccessKey secret of your Alibaba Cloud account.
  - `project_name=my_project` specifies the project that you want to access. This is the default project that you access each time you log on to the client. If `project_name` is not specified, you must run the `use project_name` command to access the project after you log on to the client.
  - Set `end_point` to the endpoint of MaxCompute. The endpoint varies based on the user account.
4. After the modification, run `./bin/odpscmd` in the Linux operating system or `./bin/odpscmd.bat` in the Windows operating system to execute SQL statements. Example:

```
create table tbl1(id bigint);
insert overwrite table tbl1 select count(*) from tbl1;
select 'welcome to MaxCompute!' from tbl1;
```

### 23.1.3. Configuration-related operations

This topic describes the configuration operations and related parameters of the MaxCompute client.

#### View the help information

Run the following command to view the help information about the client:

```
odps@ >./bin/odpscmd -h;
```

 **Note** You can also type `h;` or `help;` (not case-sensitive) in interactive mode.

## Specify startup parameters

Run the following command to specify startup parameters:

```
Usage: odpscmd [OPTION]...
where options include:
 --help (-h) for help
 --project= use project
 --endpoint= set endpoint
 -u -p user name and password
 -k will skip beginning queries and start from specified position
 -r set retry times
 -f <"file_path;"> execute command in file
 -e <"command;[command;]..."> execute command, include sql command
 -C will display job counters
```

### Example of specifying the -f parameter

1. Prepare the script.txt file that is stored in D:/. This file contains the following content:

```
drop table if exists test_table_mj;
create table test_table_mj (id string, name string);
drop table test_table_mj;
```

2. Run the following command:

```
odpscmd\bin>odpscmd -f d:/script.txt;
```

3. View the output information. Sample output:

```

ID = 20170528122432906gux77io3
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-inc.com/api&podps_public_dev&i20170528122432906gux77io3&tokenRnlrSzJoL242YW43dFFIc1dmb1ZWZzFxQ1RFPSxPRFB
TX09CTzoxMDcwMDI1NjI3ODAlNjI5LDE0MzM0MjA2NzMseyJtdGF0ZW1lbnQiO1t7IkFjdGlvbiI6WyJvZHBzO1JlYWQiXS
wiRWZmZWN0IjoiQWxs3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB_yb2plY3RzL29kcHNfcHVibGljX2Rld_i9pbnN0
YW5jZXMvMjAxNTA1MjgxmjI0MzI5MDZndXg3N2l1vMyJdfV0sIlZ1cnNpb24iOiIxIn0=
OK
ID = 20170528122439318gcmkk6u1
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-inc.com/api&podps_public_dev&i20170528122439318gcmkk6u1&tokenSt0RXdlV0M5YjZET2I1MnJuUFkzWdN1aWpzPSxPRFB
TX09CTzoxMDcwMDI1NjI3ODAlNjI5LDE0MzM0MjA2ODaseyJtdGF0ZW1lbnQiO1t7IkFjdGlvbiI6WyJvZHBzO1JlYWQiXS
wiRWZmZWN0IjoiQWxs3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB_yb2plY3RzL29kcHNfcHVibGljX2Rld_i9pbnN0Y
W5jZXMvMjAxNTA1MjgxmjI0MzI5MDZndXg3N2l1vMyJdfV0sIlZ1cnNpb24iOiIxIn0=
OK
ID = 20170528122440389g98cmlmf
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-inc.com/api&podps_public_dev&i20170528122440389g98cmlmf&tokenNW1wL0EvQThxUXhzcTRERdc5NFg0b2IxZ3QwPSxPRFB
TX09CTzoxMDcwMDI1NjI3ODAlNjI5LDE0MzM0MjA2ODaseyJtdGF0ZW1lbnQiO1t7IkFjdGlvbiI6WyJvZHBzO1JlYWQiXS
wiRWZmZWN0IjoiQWxs3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB_yb2plY3RzL29kcHNfcHVibGljX2Rld_i9pbnN0
YW5jZXMvMjAxNTA1MjgxmjI0NDazODlnOThjbWxtZiJdfV0sIlZ1cnNpb24iOiIxIn0=
OK

```

## Enter the interactive mode

You can enter the interactive mode immediately after you run the following command on the MaxCompute client.

```

[admin: ~]$odpscmd
Aliyun ODPS Command Line Tool
Version 1.0
@Copyright 2012 Alibaba Cloud Computing Co., Ltd. All rights reserved.
XXX@ XXX> insert overwrite table dual select * from dual;

```

**Note** The first XXX indicates the identifier of MaxCompute, and the second XXX indicates the project to which you belong. Enter a command that is terminated by a semicolon (;) at the cursor, and press Enter to run the command.

## Specify the output format

The output of an SQL statement is in the human-readable or machine-readable format. Human-readable is the default format. If you use the `-M` parameter when you run `odpscmd`, the output format is CSV.

**Note** This feature applies only to SELECT and READ statements and takes effect during data reading.

## Specify the statements to execute

If you specify the `-e` or `-f` parameter and want to start with an intermediate statement among a few statements, you can specify the `-k` parameter. The `-k` parameter indicates that the execution starts from the specified statement and the preceding statements are skipped. If the value of the `-k` parameter is less than or equal to 0, the execution starts from the first statement. A statement terminated by a semicolon (;) is considered valid. At runtime, the output information indicates the specific statement that is executed or failed.

For example, the `dual.sql` file in the `tmp` folder contains the following SQL statements:

```
drop table dual;
create table dual (dummy string);
insert overwrite table dual select count(*) from dual;
```

You can run the following command to skip the first two statements:

```
odpscmd -k 3 -f dual.sql
```

## Obtain information about the current logon user

Run a command to obtain the Alibaba Cloud account of the current logon user and the endpoint that is used.

Command syntax:

```
whoami
```

Sample command:

```
odps@ hiveut>whoami; Name: odpstest@aliyun.com ID: 1090142773636588 End_Point: <Endpoint of MaxCompute> Project: lijunsecuritytest
```

## Exit the MaxCompute client

Command syntax:

```
odps@ > quit;
```

You can also run the following command:

```
q;
```

## Configure a job priority

Command syntax:

```
Admin@ > ./bin/odpscmd --instance-priority=<PRIORITY>;
```

Configuration file: odps\_config.ini

```
instance_priority=<PRIORITY>
```

### Notice

- <PRIORITY >
- The priority setting in the configuration file applies to all instances submitted in the MaxCompute client.
- If you do not configure the priority for a job, its priority is 9.

## Dry run mode

In dry run mode, MaxCompute parses an SQL statement to check whether the syntax of this statement is correct and generates an execution plan. In this mode, MaxCompute does not submit a distributed job. Command syntax:

```
./bin/odpscmd -y
```

## SQL reliability

If you execute statements, such as `INSERT` or `CREATE TABLE AS`, on the MaxCompute client and an exception occurs when you run a job, the MaxCompute client automatically restores data and metadata to the status before the SQL query is run based on the known information. The exceptions include:

- Data that is overwritten by using the `INSERT OVERWRITE` statement during queries is restored from the temporary backup directory to the original directory.
- Data that is generated by using the `INSERT INTO` statement during queries is deleted.
- Tables that are created during queries or the dynamically generated partition information that does not exist before queries are deleted.

If MaxCompute fails to restore data, the following error code is returned. The error code informs you that further attempts may cause some data unrecoverable. Error:

```
ODPS-
0110999: Critical! Internal error happened in commit operation and rollback failed, possible breach
of atomicity
```

# 23.2. Eclipse development plugin

## 23.2.1. Install the Eclipse development plug-in

This topic describes how to install the Eclipse development plug-in.

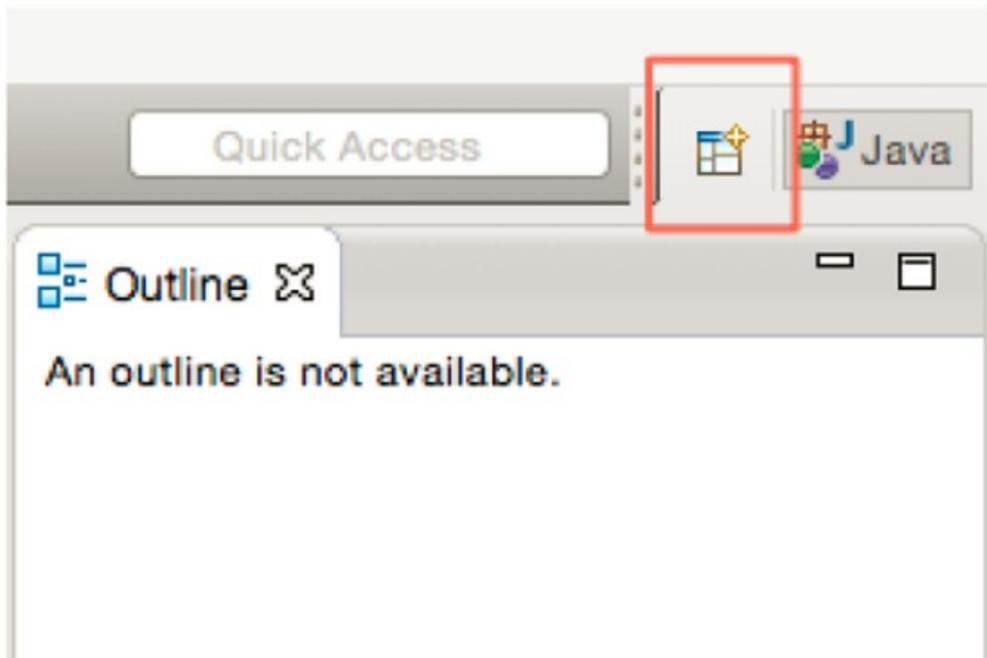
### Context

MaxCompute provides the Eclipse development plug-in to help you use SDKs for Java to develop MapReduce programs or user-defined functions (UDFs). This plug-in can simulate the running process of MapReduce programs or UDFs. It provides local debugging methods and features to generate templates. You can click [Eclipse](#) to download the plug-in package.

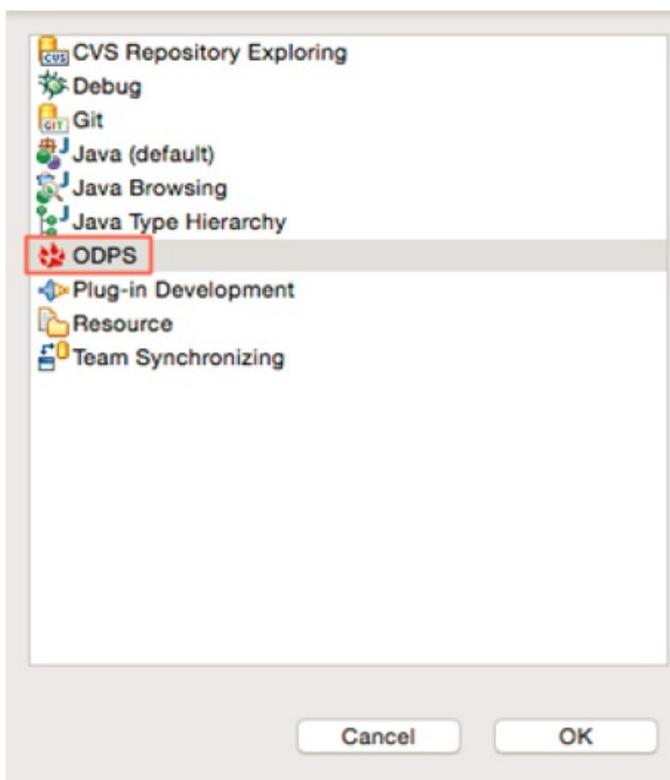
 **Note** Unlike the MapReduce program that runs in local mode, the Eclipse development plug-in cannot synchronize data with MaxCompute. You must manually copy the required data to the *warehouse* directory of the Eclipse development plug-in.

### Procedure

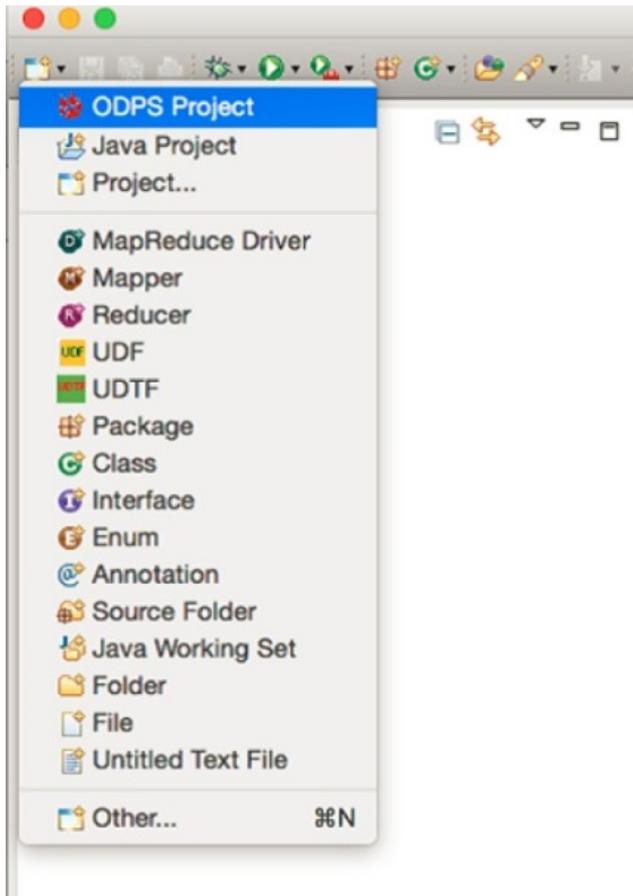
1. Decompress the Eclipse package to view the following JAR file in the package:  
*odps-eclipse-plugin-bundle-0.16.0.jar*
2. Place the JAR file in the *plugins* folder under the Eclipse installation directory.
3. Start the Eclipse development plug-in and click the **Open Perspective** icon in the upper-right corner, as shown in the following figure.



4. In the dialog box that appears, click **ODPS** and then **OK**.



5. In the navigation bar that appears, select the **ODPS project** and click **OK**. The **ODPS icon** is displayed in the upper-right corner, as shown in the following figure. The icon indicates that the plug-in has taken effect.



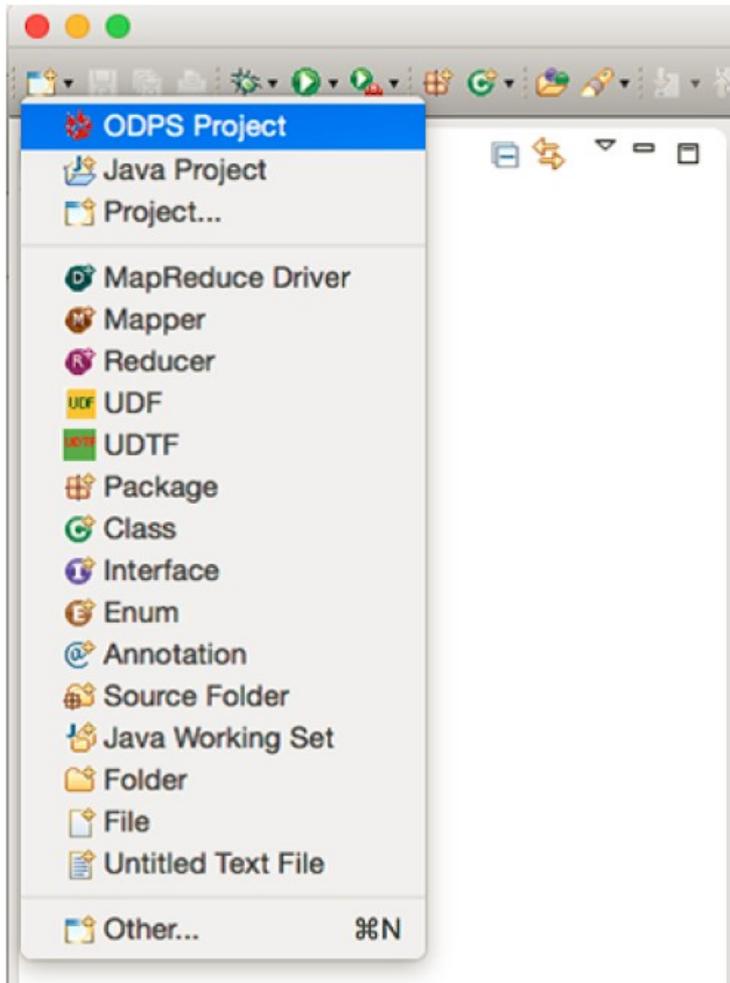
## 23.2.2. Create a project

### 23.2.2.1. Method 1

This topic describes Method 1. This method is used to create a project in the Eclipse development plug-in.

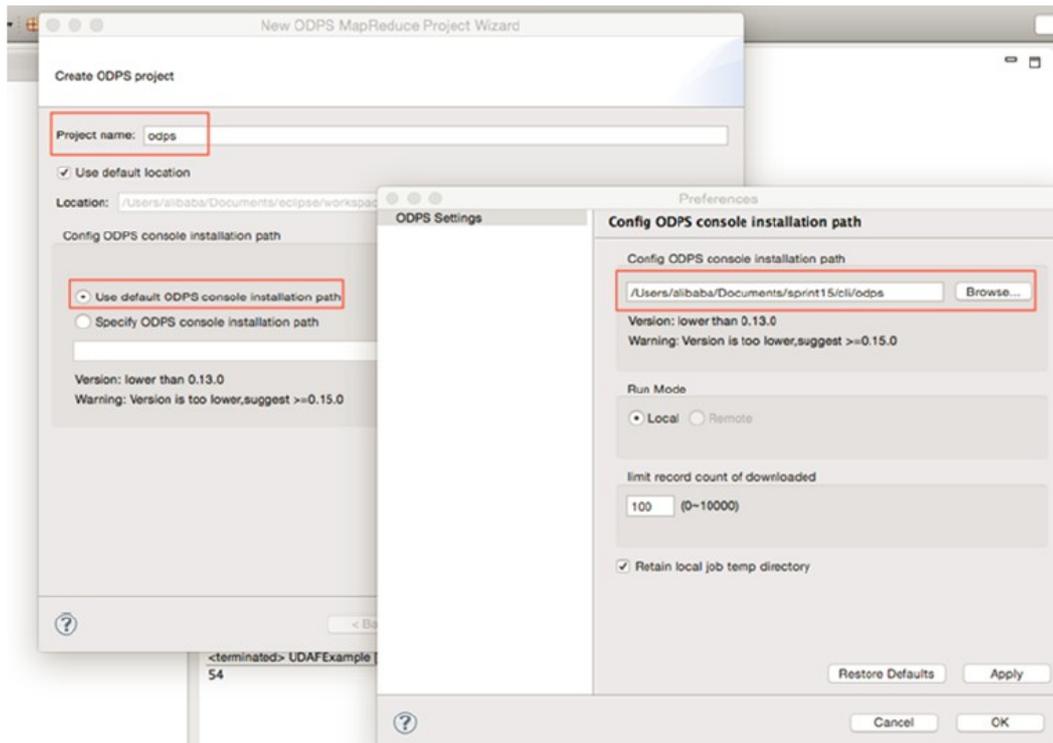
#### Procedure

1. In the upper-left corner of Eclipse, choose **File > New > Project > ODPS > ODPS Project**, as shown in the following figure.



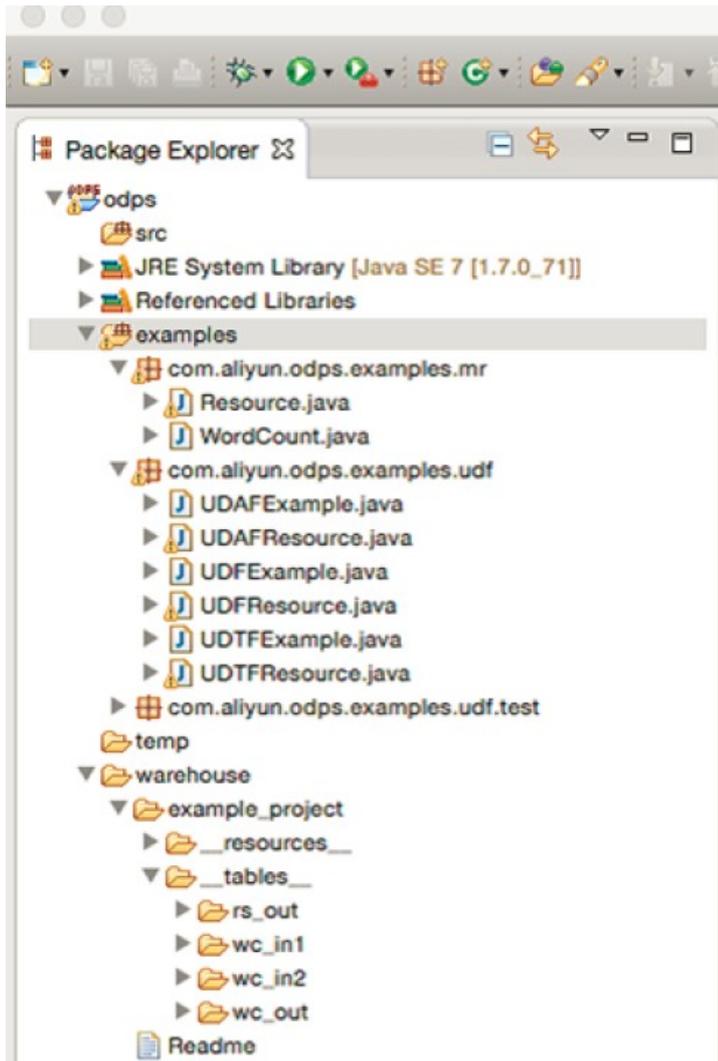
 **Note** In this example, the project name is ODPS.

2. In the dialog box shown in the following figure, enter the project name, select the installation path of the MaxCompute client, and then click **Finish**.



 **Note** The client must be installed in advance.

3. After you create a project, you can view the directory structure on the Package Explorer tab, as shown in the following figure.

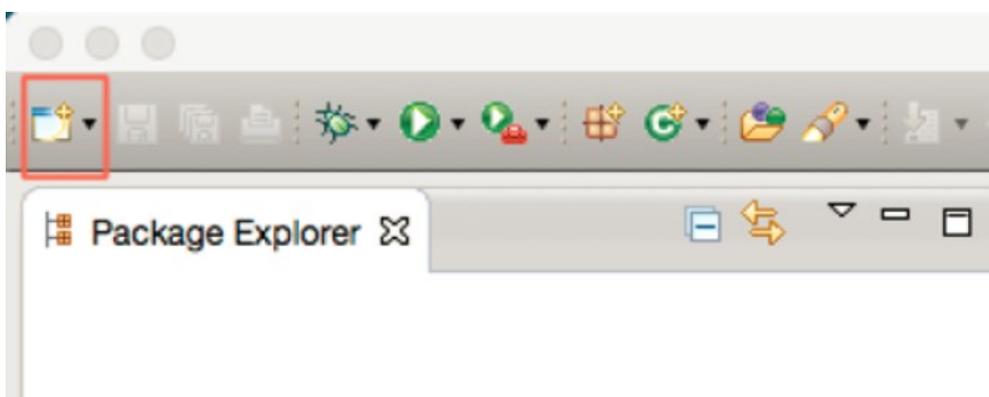


### 23.2.2.2. Method 2

This topic describes another method to create a project by using the Eclipse plug-in.

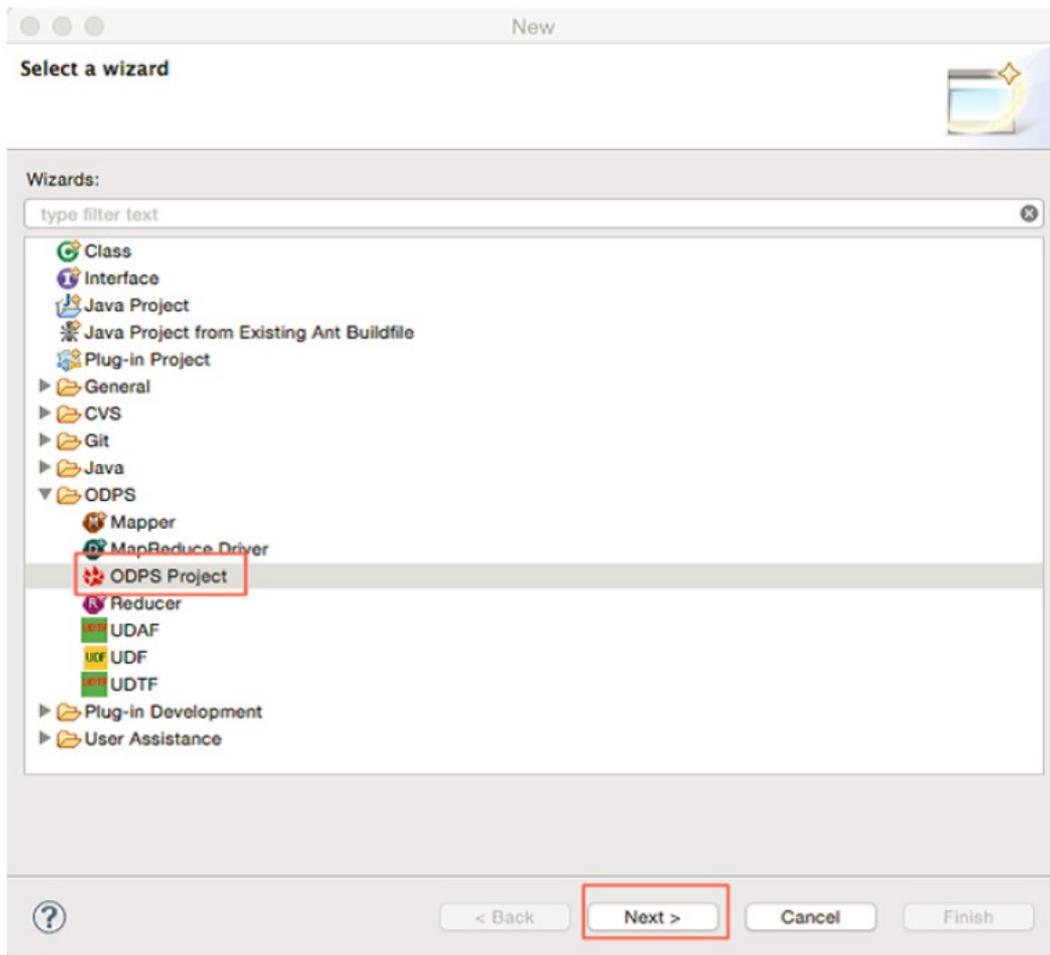
#### Procedure

1. In the toolbar of the Eclipse plug-in, click the **New** icon, as shown in the following figure.



2. In the dialog box that appears, unfold ODPS, click **ODPS Project**, and then click **Next**, as shown in the

following figure.



**Note** In this example, the project name is ODPS.

- The subsequent steps are the same as those in method 1. After you install the Eclipse plug-in, you can use it to compile a MapReduce program or user-defined function (UDF).

**Note** For more information about how to run a MapReduce program by using the Eclipse plug-in, see [MapReduce running example](#). For more information about how to develop and run a UDF, see [UDF development and running example](#).

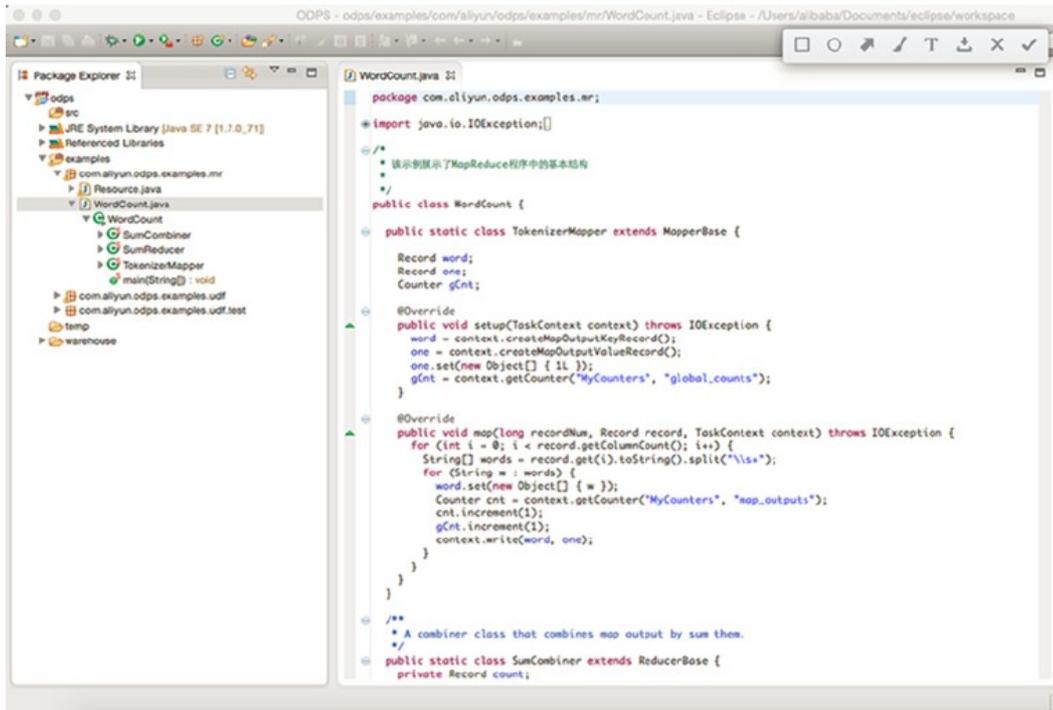
## 23.2.3. MapReduce running example

### 23.2.3.1. Run a WordCount program

This topic describes how to use MapReduce to run a WordCount program in the Eclipse development plug-in.

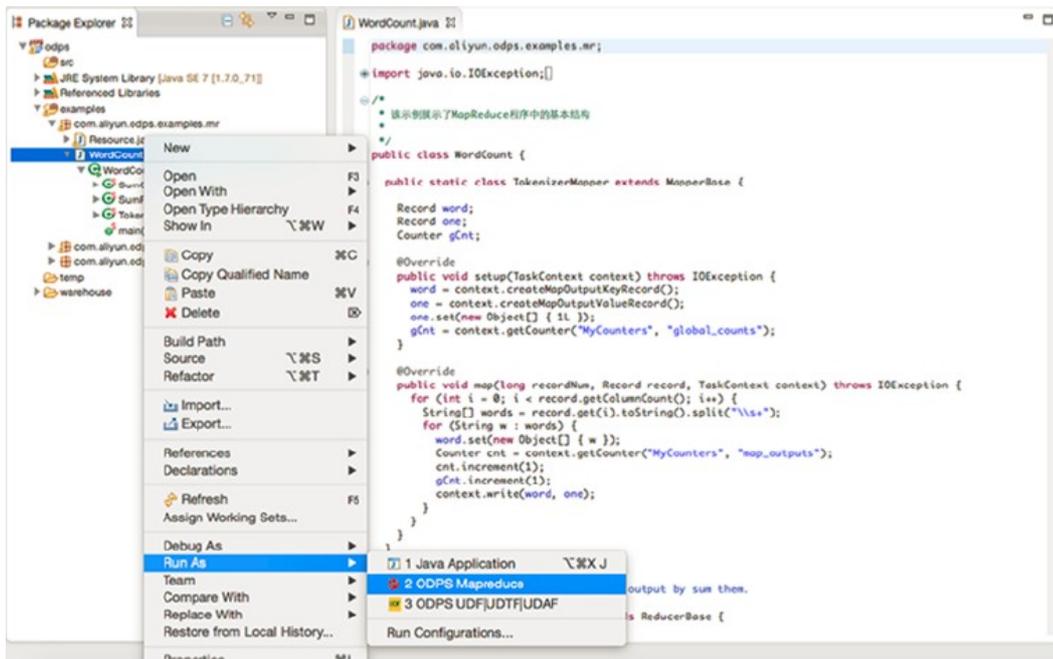
#### Procedure

- On the Package Explorer tab of the Eclipse development plug-in, choose Examples > com.aliyun.odps.examples.mr > WordCount.java from the navigation tree, as shown in the following figure.  
Choose the WordCount program



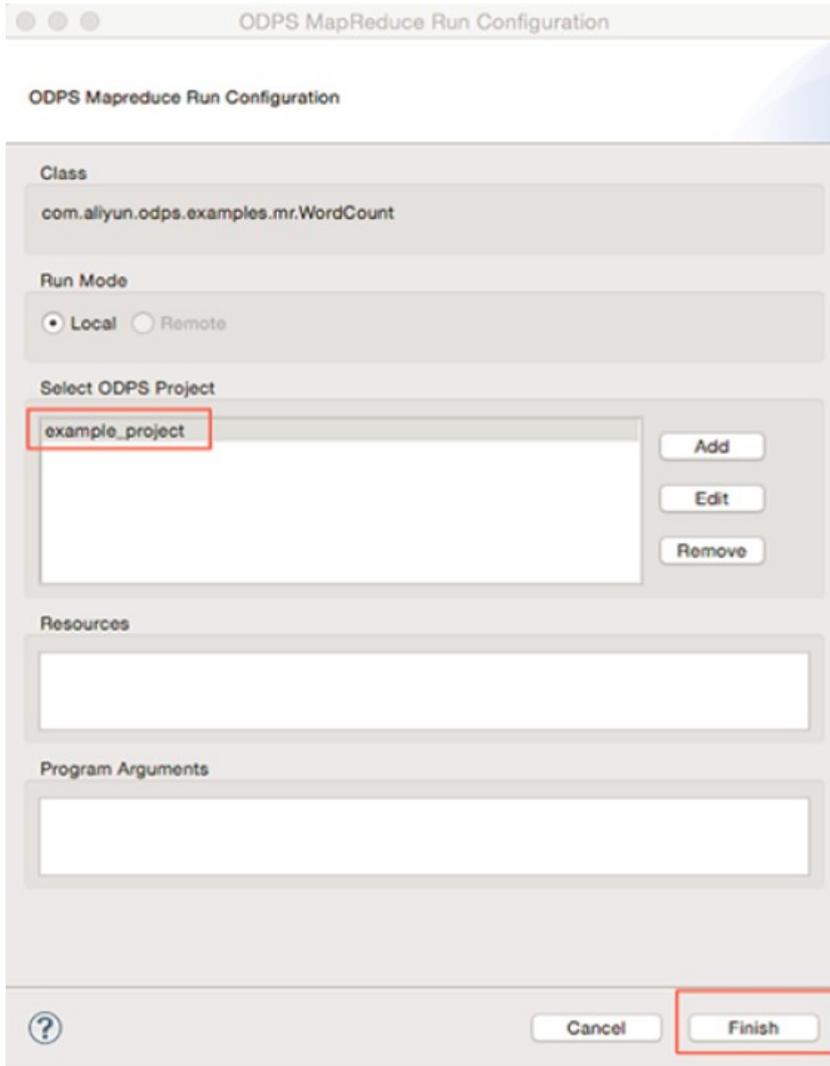
2. Right-click the **WordCount.java** program and choose **Run As2 ODPS Mapreduce**, as shown in the following figure.

Run the WordCount program (1)



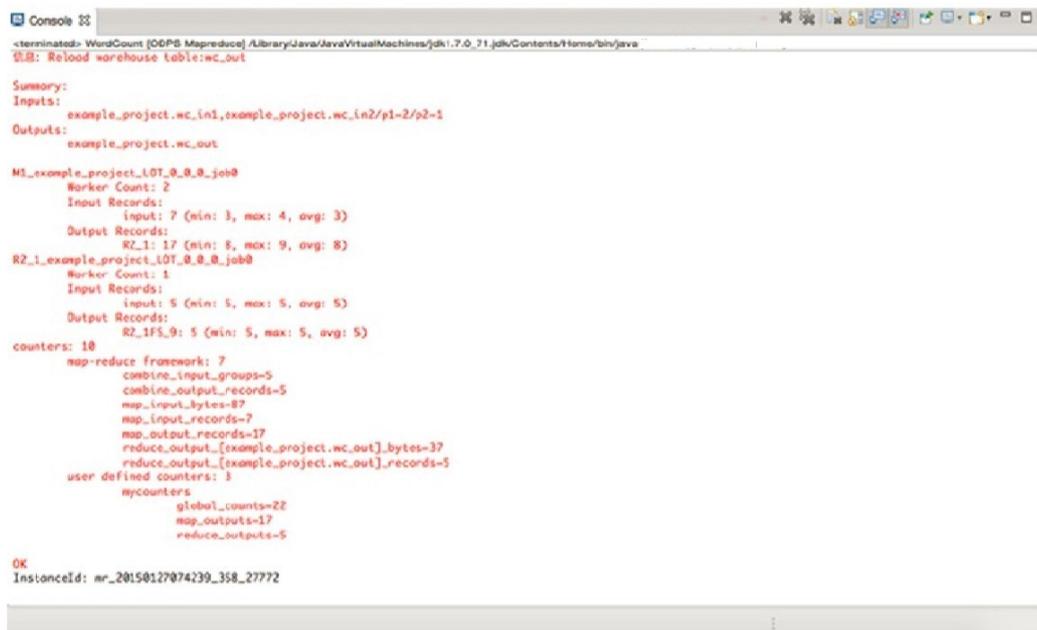
3. In the ODPS MapReduce Run Configuration dialog box, add **example\_project** to the Select ODPS Project section and click **Finish**, as shown in the following figure.

Run the WordCount program (2)



4. After you run the WordCount program, the result is displayed, as shown in the following figure.

Execution result



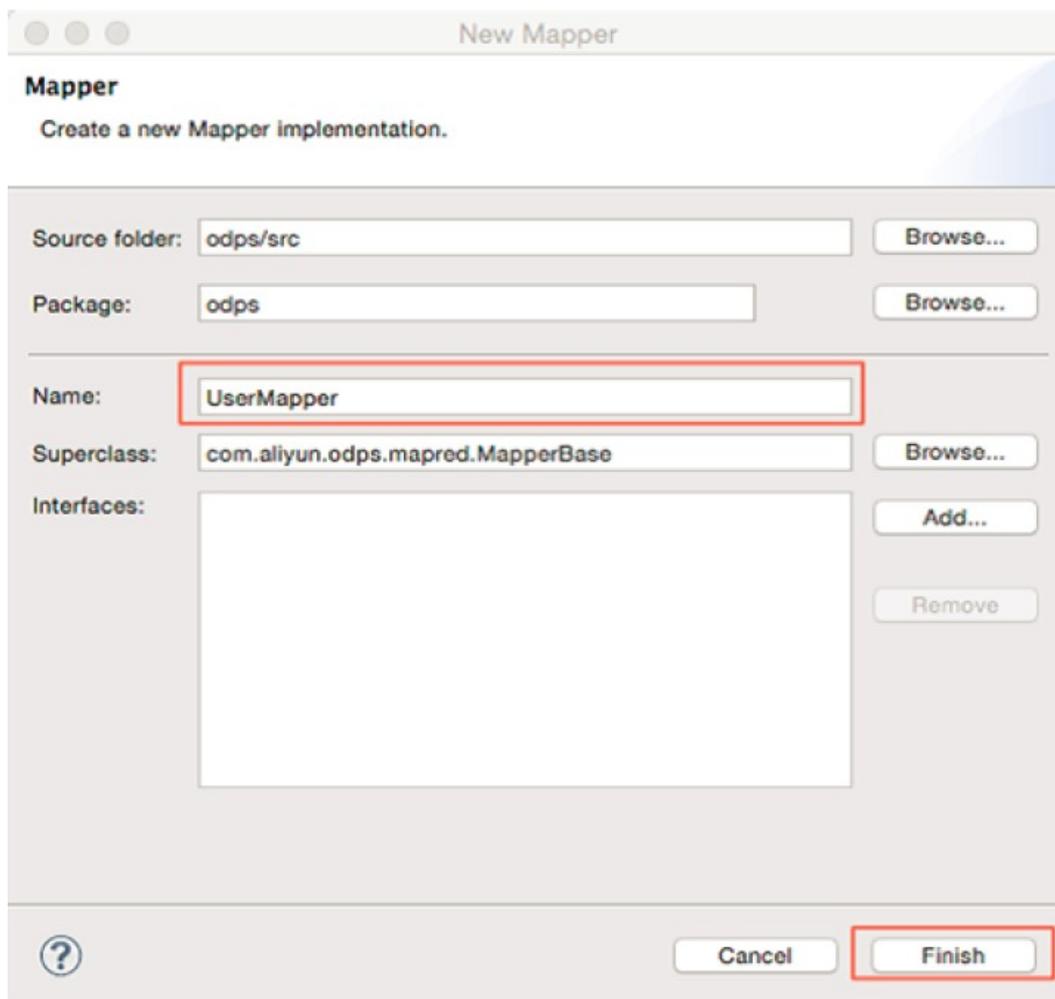
### 23.2.3.2. Run a custom MapReduce program

This topic provides an example on how to run a custom MapReduce program by using the Eclipse development plug-in.

#### Procedure

1. On the Package Explorer tab of the Eclipse development plug-in, choose `odps > src` from the navigation tree. Right-click `src` and choose **New > Mapper**.
2. In the New Mapper dialog box, enter the name of the Mapper class and click **Finish**, as shown in the following figure.

New Mapper dialog box

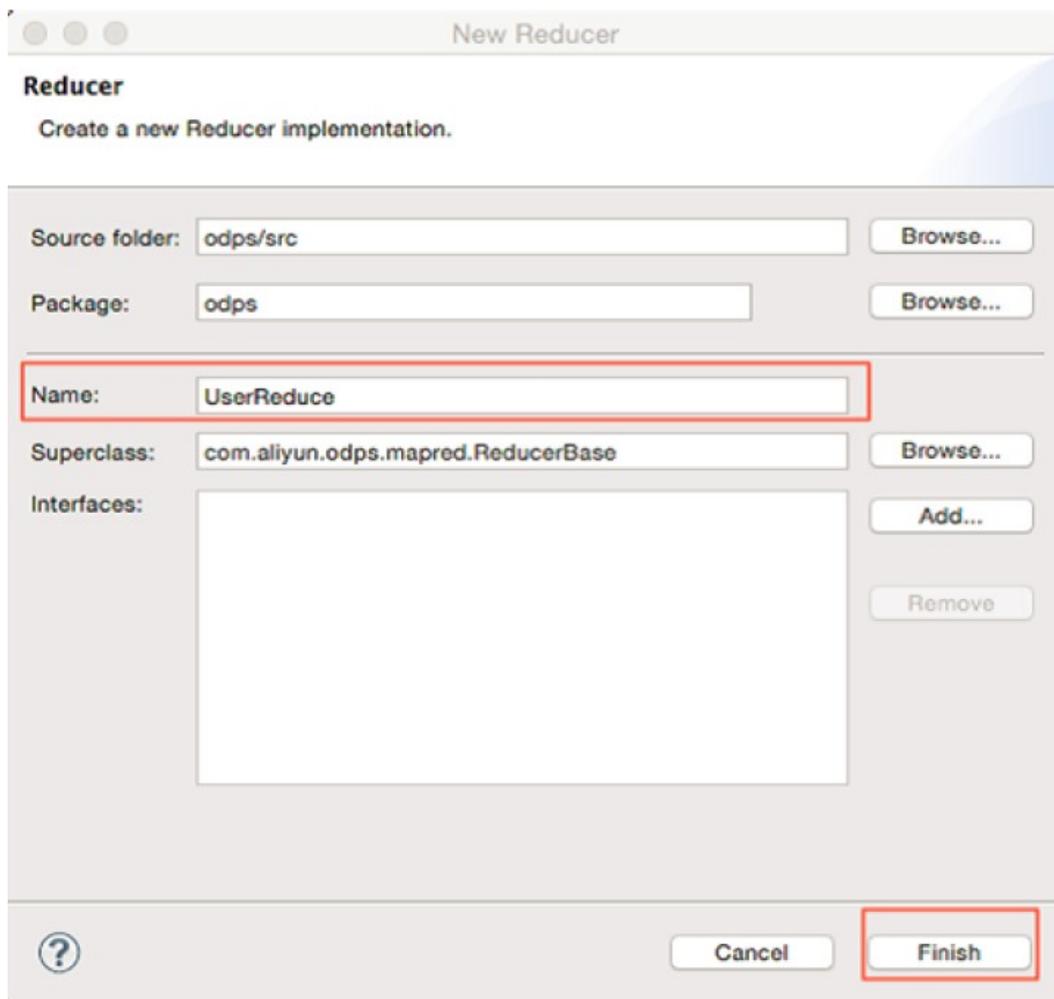


3. The `UserMapper.java` file is generated in the `src` folder shown in the navigation tree on the Package Explorer tab. The file is a Mapper class template. By default, the package name in the template is `odps`. The template contains the following information:

```
package odps;
import java.io.IOException;
import com.aliyun.odps.counter.Counter; import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.MapperBase;
public class UserMapper extends MapperBase {
Record word; Record one; Counter gCnt;
@Override
public void setup(TaskContext context) throws IOException {
word = context.createMapOutputKeyRecord(); one = context.createMapOutputValueRecord(); one.set(new Object[] { 1L });
gCnt = context.getCounter("MyCounters", "global_counts");
}
@Override
public void map(long recordNum, Record record, TaskContext context) throws IOException {
for (int i = 0; i < record.getColumnCount(); i++) { String[] words = record.get(i).toString().split("\\s+"); for (String w : words) {
word.set(new Object[] { w });
Counter cnt = context.getCounter("MyCounters", "map_outputs"); cnt.increment(1);
gCnt.increment(1); context.write(word, one);
}
}
}
@Override
public void cleanup(TaskContext context) throws IOException {
}
}
```

4. On the Package Explorer tab of the Eclipse development plug-in, choose `odps > src` from the navigation tree. Right-click `src` and choose **New > Reduce**.
5. In the New Reducer dialog box, enter the name of the Reducer class and click **Finish**, as shown in the following figure.

Reducer



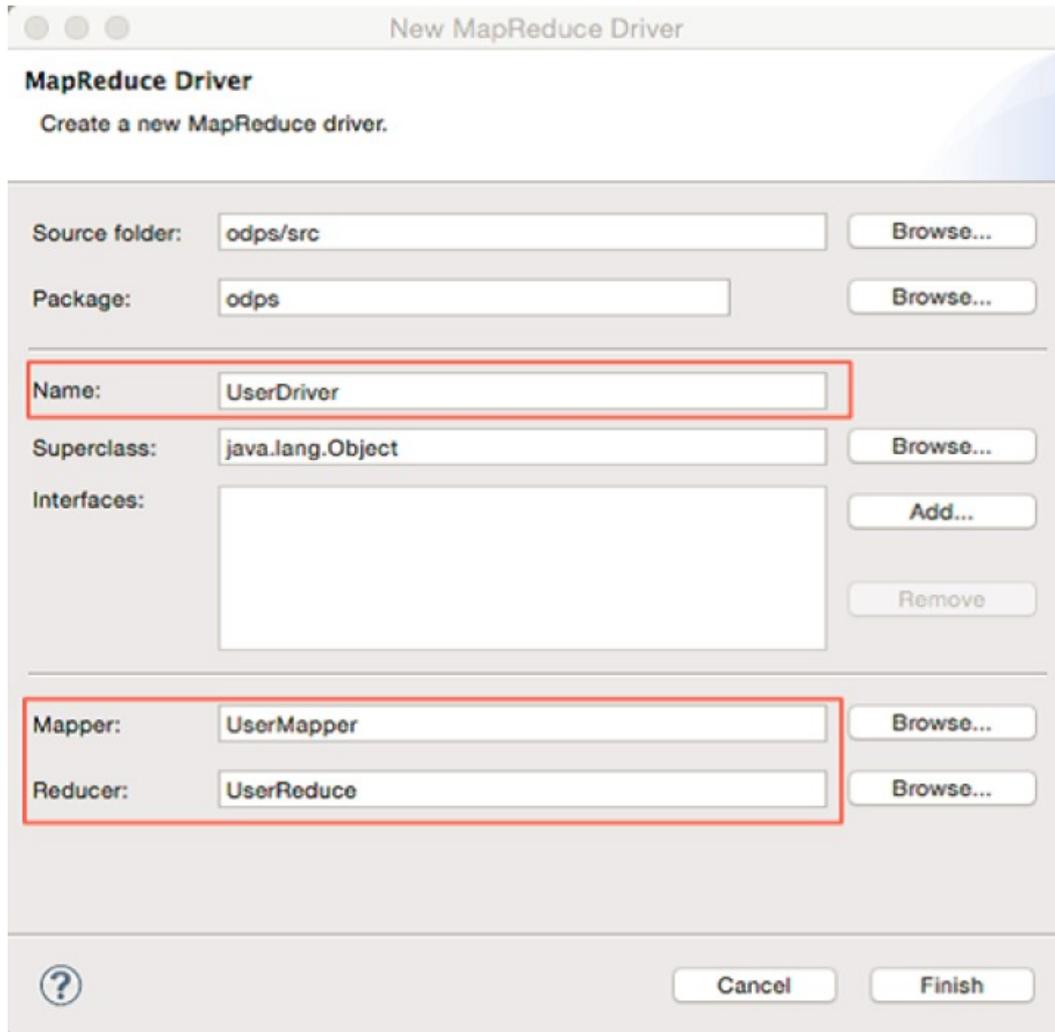
 **Note** In this example, the name of the Reducer class is UserReducer.

6. On the Package Explorer tab, the UserReducer.java file is generated in the src folder. The file is a Reducer class template. By default, the package name in the template is odps. The template contains the following information:

```
package odps;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.ReducerBase;
public class UserReducer extends ReducerBase {
private Record result; Counter gCnt;
@Override
public void setup(TaskContext context) throws IOException { result = context.createOutputRecord(
);
gCnt = context.getCounter("MyCounters", "global_counts");
}
@Override
public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException
{
long count = 0;
while (values.hasNext()) { Record val = values.next(); count += (Long) val.get(0);
}
result.set(0, key.get(0)); result.set(1, count);
Counter cnt = context.getCounter("MyCounters", "reduce_outputs"); cnt.increment(1);
gCnt.increment(1);
context.write(result);
}
@Override
public void cleanup(TaskContext context) throws IOException {
}
}
```

7. On the Package Explorer tab of the Eclipse development plug-in, choose odps > src from the navigation tree. Right-click src and choose **New > MapReduce Driver**.
8. In the New MapReduce Driver dialog box, Specify Name, Mapper, and Reducer, and click **Finish**, as shown in the following figure.

MapReduce Driver

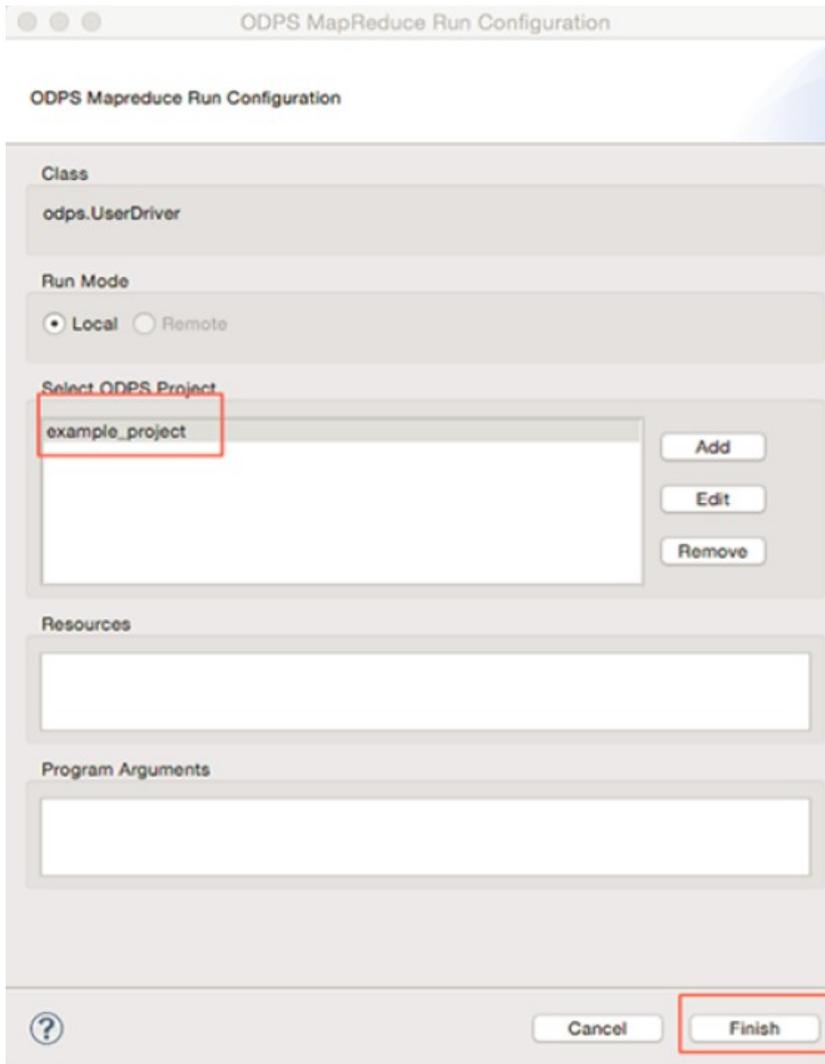


9. On the Package Explorer tab, the MyDriver.java file is generated in the src folder. The file is a MapReduce Driver class template. By default, the package name in the template is odps. The template contains the following information:

```
package odps;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.examples.mr.WordCount.SumCombiner;
import com.aliyun.odps.examples.mr.WordCount.SumReducer;
import com.aliyun.odps.examples.mr.WordCount.TokenizerMapper;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
public class UserDriver {
public static void main(String[] args) throws OdpsException { JobConf job = new JobConf();
job.setMapperClass (TokenizerMapper.class);
job.setCombinerClass (SumCombiner.class);
job.setReducerClass (SumReducer.class);
job.setMapOutputKeySchema (SchemaUtils.fromString("word:string"));
job.setMapOutputValueSchema (SchemaUtils.fromString("count:bigint"));
InputUtils.addTable (
TableInfo.builder().tableName("wc_in1").cols(new String[] { "col2", "col3" }).build(), job);
InputUtils.addTable (TableInfo.builder().tableName("wc_in2").partSpec("p1=2/p2=1").build(), job);
OutputUtils.addTable (TableInfo.builder().tableName("wc_out").build(), job);
RunningJob rj = JobClient.runJob(job); rj.waitForCompletion();
}
}
```

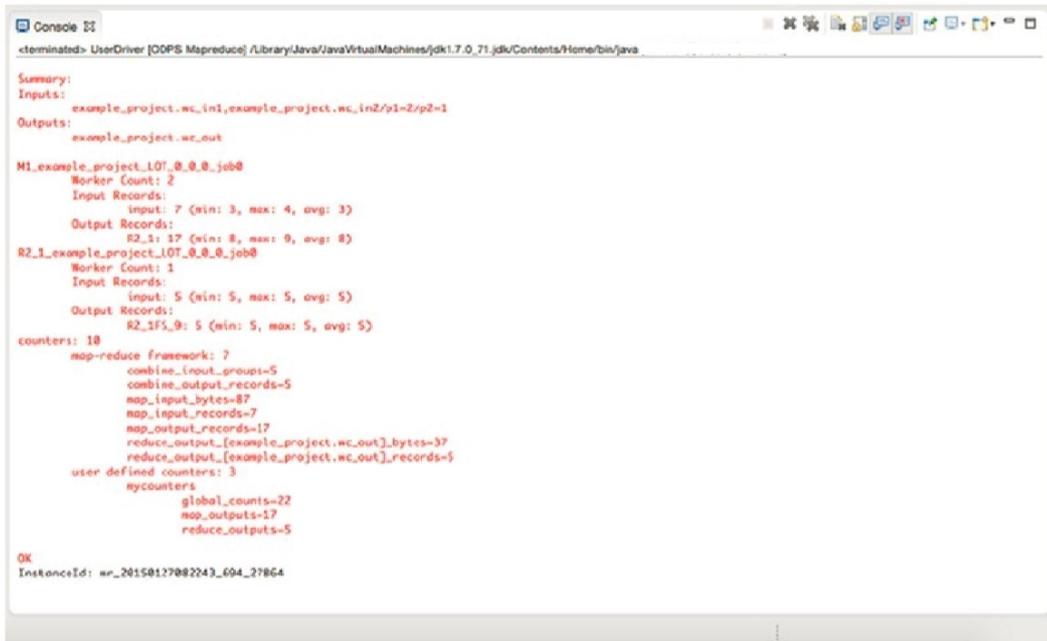
10. Run the MapReduce program. Right-click the UserDriver.java file, choose **Run As > ODPS MapReduce**, and then click **OK**. The ODPS MapReduce Run Configuration dialog box is displayed, as shown in the following figure.

#### ODPS MapReduce Run Configuration



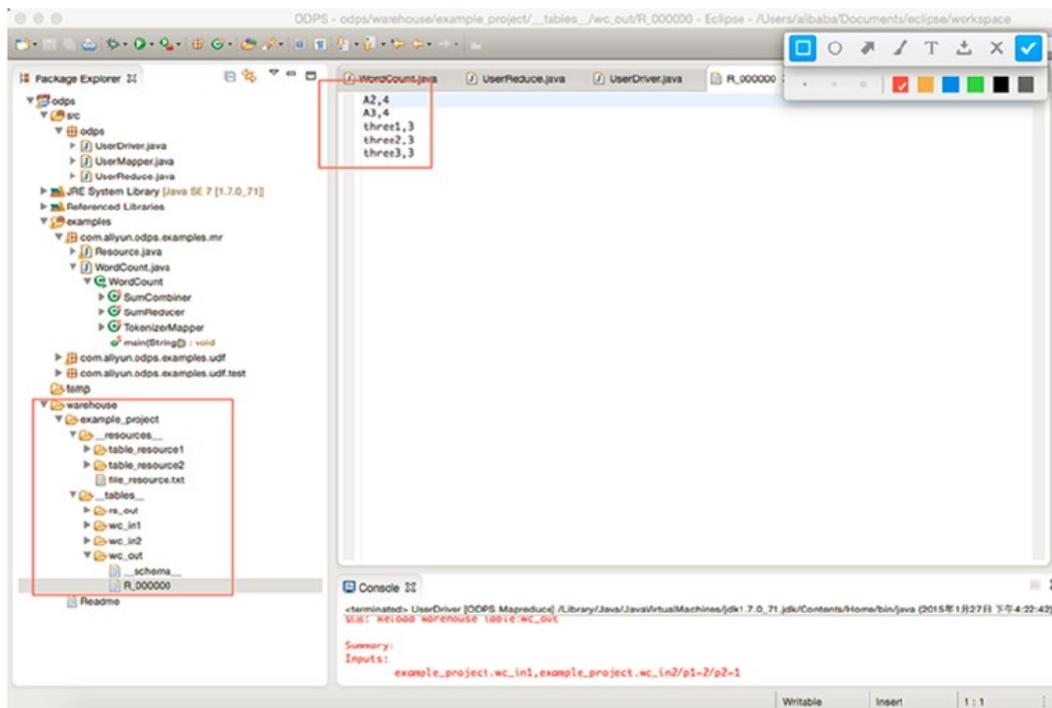
11. Add `example_project` in Select ODPS Project and click **Finish** to run the MapReduce program in local mode. If the following information is displayed, the MapReduce program properly runs in local mode.

Console



- 12. Right-click the *warehouse* folder in the navigation tree on the Package Explorer tab and select Refresh to view the output result, as shown in the following figure.

Output result

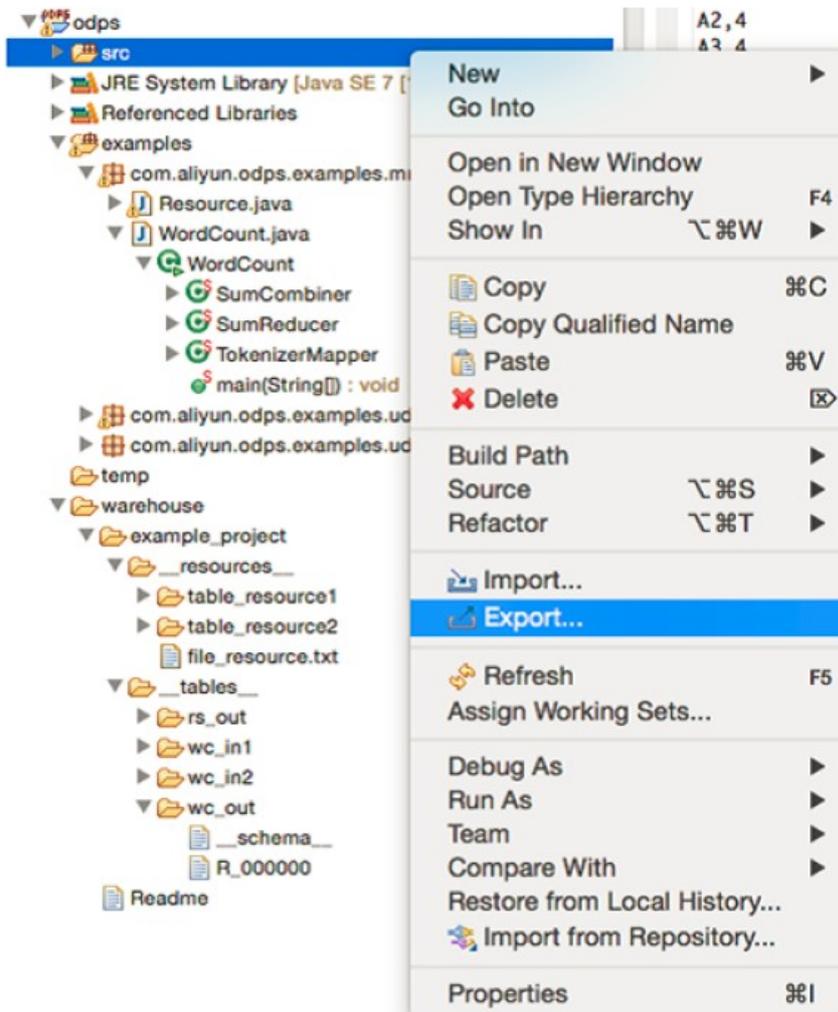


ⓘ Note wc\_out is the output folder, and R\_000000 is the result file. After you perform local debugging to verify that the output result is correct, you can use the export feature of the Eclipse development plug-in to package the MapReduce program for future use in a distributed environment.

- 13. Export the MapReduce program package.

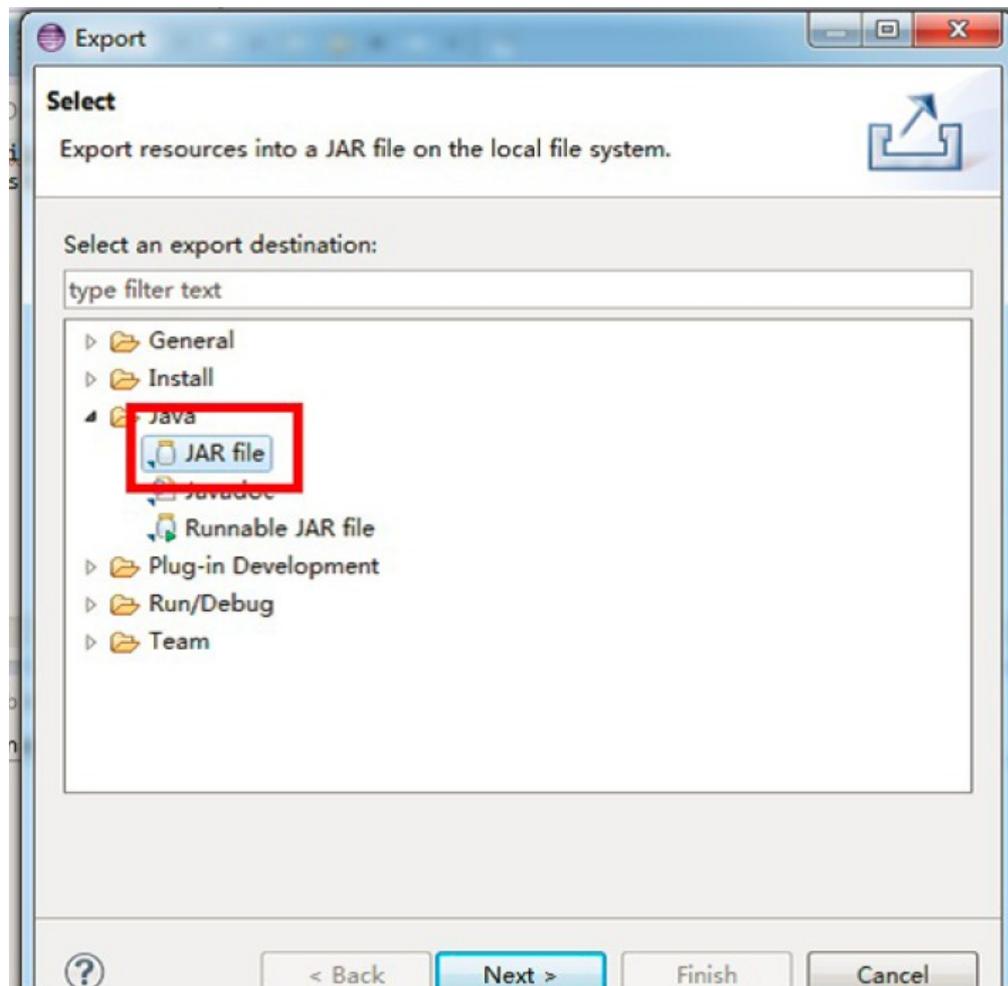
- i. On the Package Explorer tab of the Eclipse development plug-in, choose odps > src from the navigation tree. Right-click src and select **Export**, as shown in the following figure.

Export dialog box



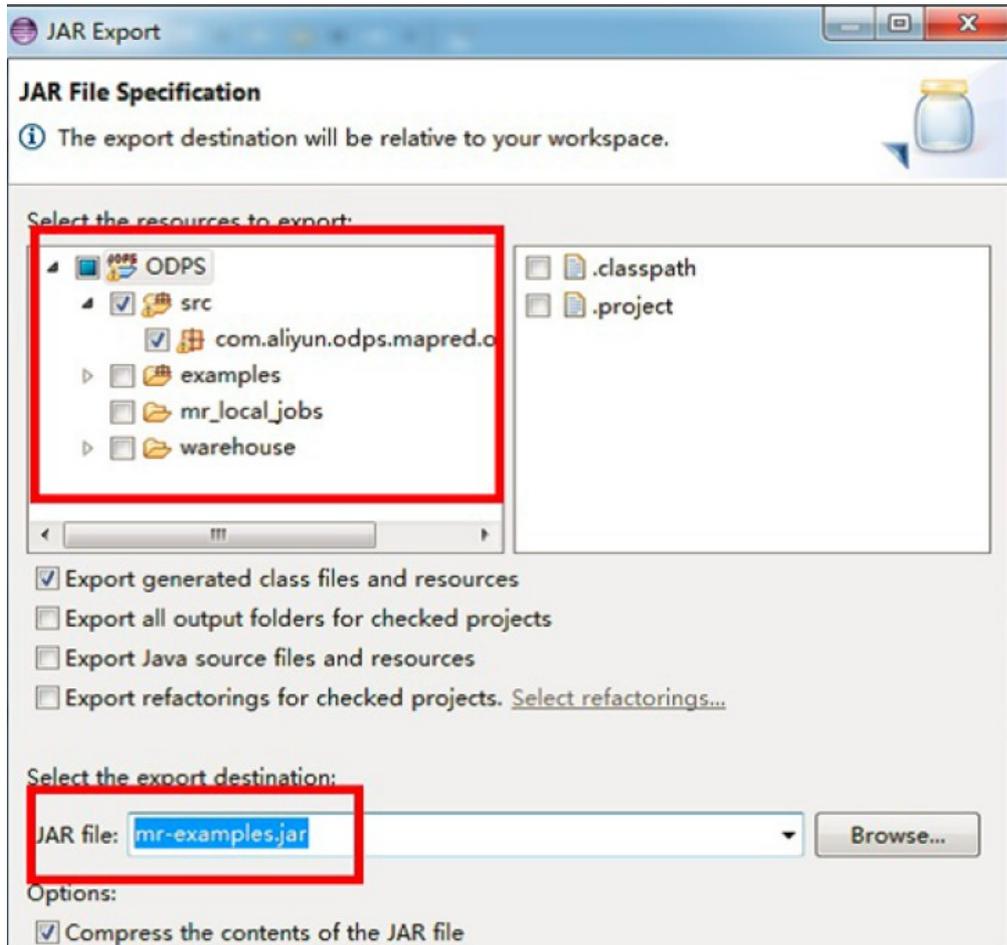
- ii. In the Export dialog box, choose Java > JAR file in Select an export destination and click Next, as shown in the following figure.

JAR Export dialog box



- iii. In the JAR Export dialog box, choose ODPS > src > com.aliyun.odps.mapred.open.example, and specify the mr-examples.jar file as the MapReduce program package, as shown in the following figure.

Select the source and destination packages



**Note** In this example, the name of the MapReduce program package is mr-examples.jar. You can name the package based on your business requirements.

- iv. Click Finish. The export process is complete.
14. If you want to create a project, you can create a folder at the same directory level as the example\_project folder in the warehouse folder. The following figure shows the directory structure.

```

<warehouse>
 |__example_project
 |__<_tables_>
 | |__table_name1
 | | |__data
 | | |
 | | |__<_schema_>
 | |
 | |__table_name2
 | | |__partition_name=partition_value
 | | | |__data

 | |
 | |__<_schema_>
 |
 |__<_resources_>
 |
 |__table_resource_name
 | |__<_ref_>
 |
 |__file_resource_name

```

Examples of information in schema files:

```

project=project_name table=table_name
columns=col1:BIGINT,col2:DOUBLE,col3:BOOLEAN,col4:DATETIME,col5:STRING
-- Information in the schema file of a non-partitioned table:
project=project_name table=table_name
columns=col1:BIGINT,col2:DOUBLE,col3:BOOLEAN,col4:DATETIME,col5:STRING partitions=col1:BIGINT,col2:DOUBLE,col3:BOOLEAN,col4:DATETIME,col5:STRING
-- Information in the schema file of a partitioned table:
-- The following data types are supported: BIGINT, DOUBLE, BOOLEAN, DATETIME, and STRING. These data types correspond to LONG, DOUBLE, BOOLEAN, Java.Util.Date, and Java.Lang.STRING in Java.

```

Example of a data file:

```

1,1.1,true,2015-06-04 11:22:42 896,hello world
\n,\n,\n,\n,\n
-- The time is accurate to milliseconds. \n represents NULL for all data types.

```

**Note**

- Before the MapReduce program runs in local mode, the program automatically checks for data tables or resources in the warehouse folder first. If the tables or resources are not found, the program downloads the tables or resources from a server to the *warehouse* folder.
- After you run the MapReduce program, right-click the *warehouse* folder in the navigation tree on the Package Explorer tab and select Refresh to view the output result.

## 23.2.4. UDF development and running example

### 23.2.4.1. Local debug UDF programs

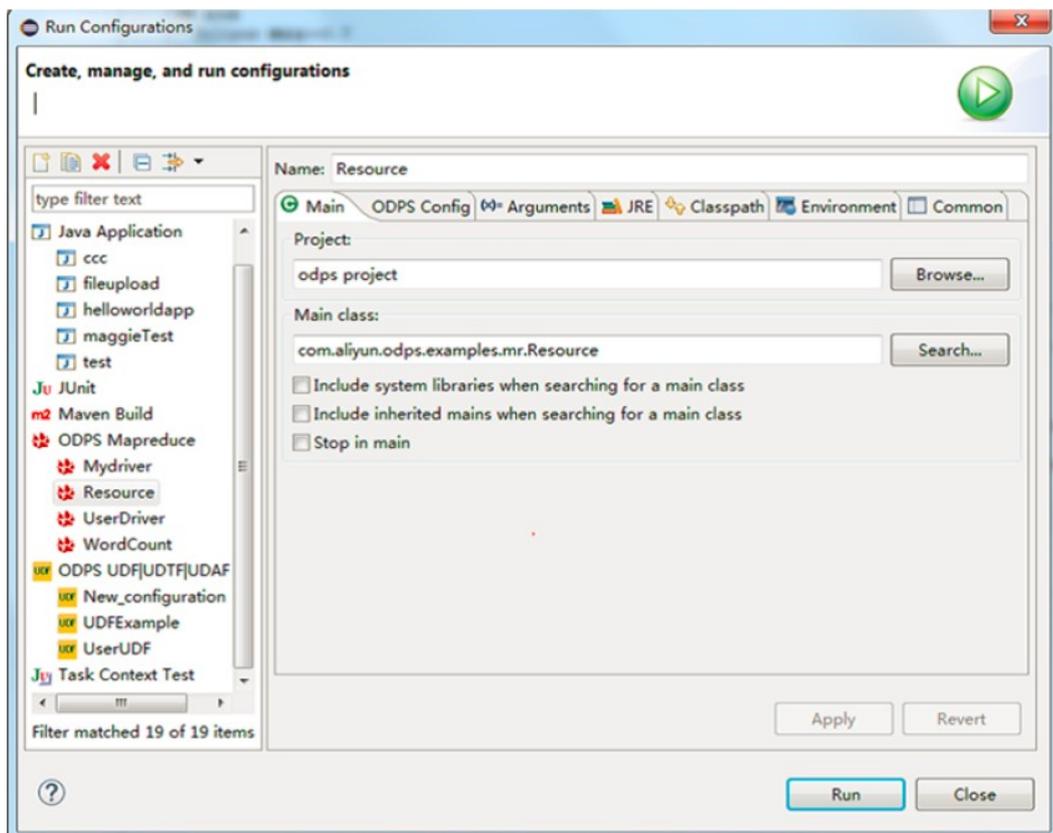
#### 23.2.4.1.1. Run a UDF from the menu bar

This topic describes how to run a user-defined function (UDF) from the menu bar of the Eclipse development plug-in.

#### Procedure

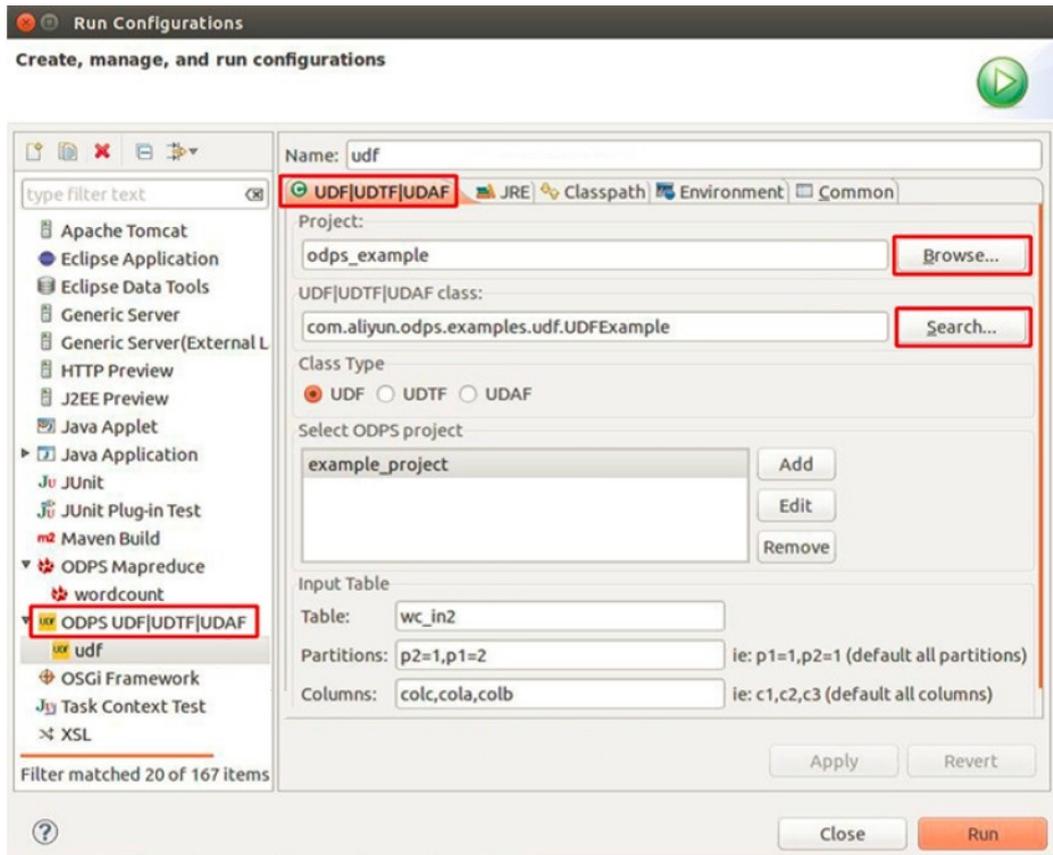
1. Choose **Run > Run Configurations** from the menu bar. The Run Configurations dialog box appears, as shown in the following figure.

Run Configurations



2. In the left-side navigation tree, right-click ODPS UDF|UDTF|UDAF and select New. On the ODPS UDF|UDTF|UDAF tab, select the project, UDF class, class type, and ODPS project, and enter the table information, as shown in the following figure.

Run Configurations 2



Note In the preceding figure, the Table parameter indicates the input table of the UDF. The Partitions parameter indicates the partitions from which you want to read data. Partition names are separated by commas (.). The Columns parameter indicates the columns that are passed as parameters of the UDF. Column names are separated by commas (.).

3. Click Run. The running result is displayed in the console, as shown in the following figure.

Console



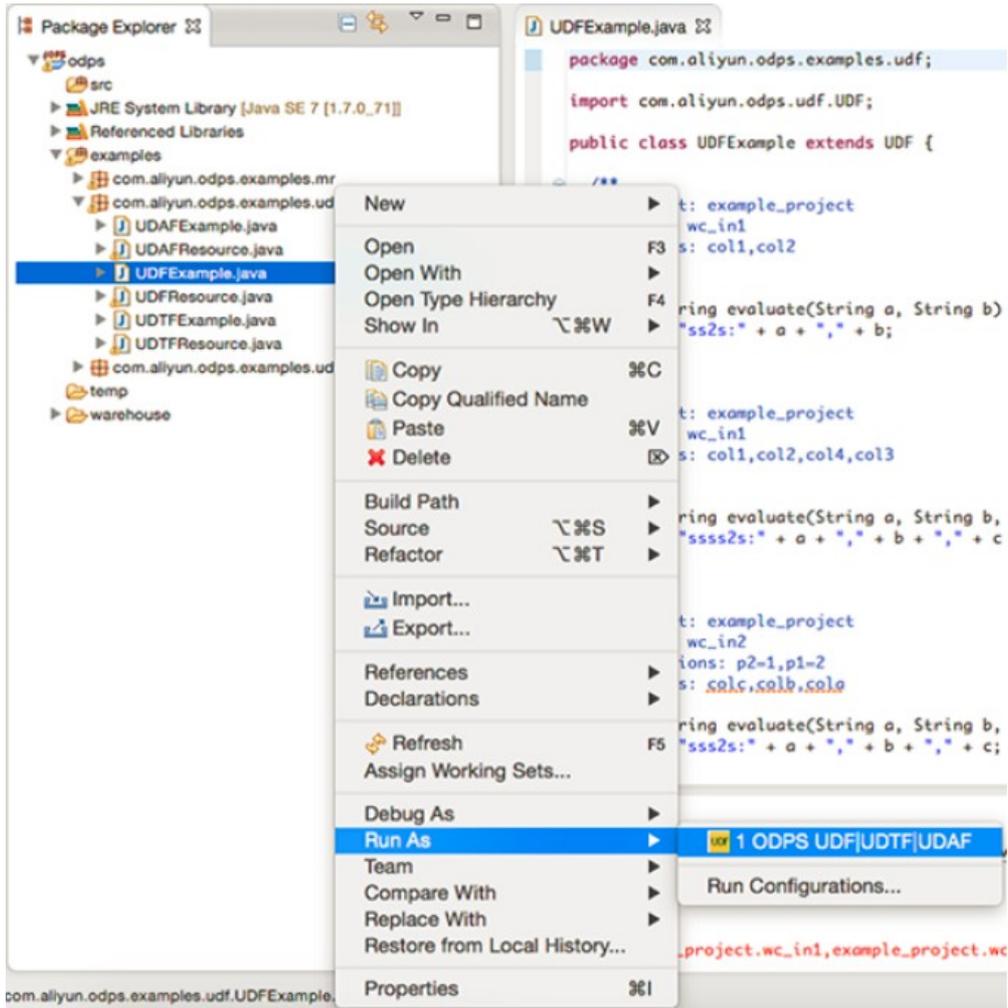
### 23.2.4.1.2. Use the right-click shortcut menu to run a UDF

This topic describes how to use the right-click shortcut menu to run a user-defined function (UDF) in the Eclipse development plug-in.

#### Procedure

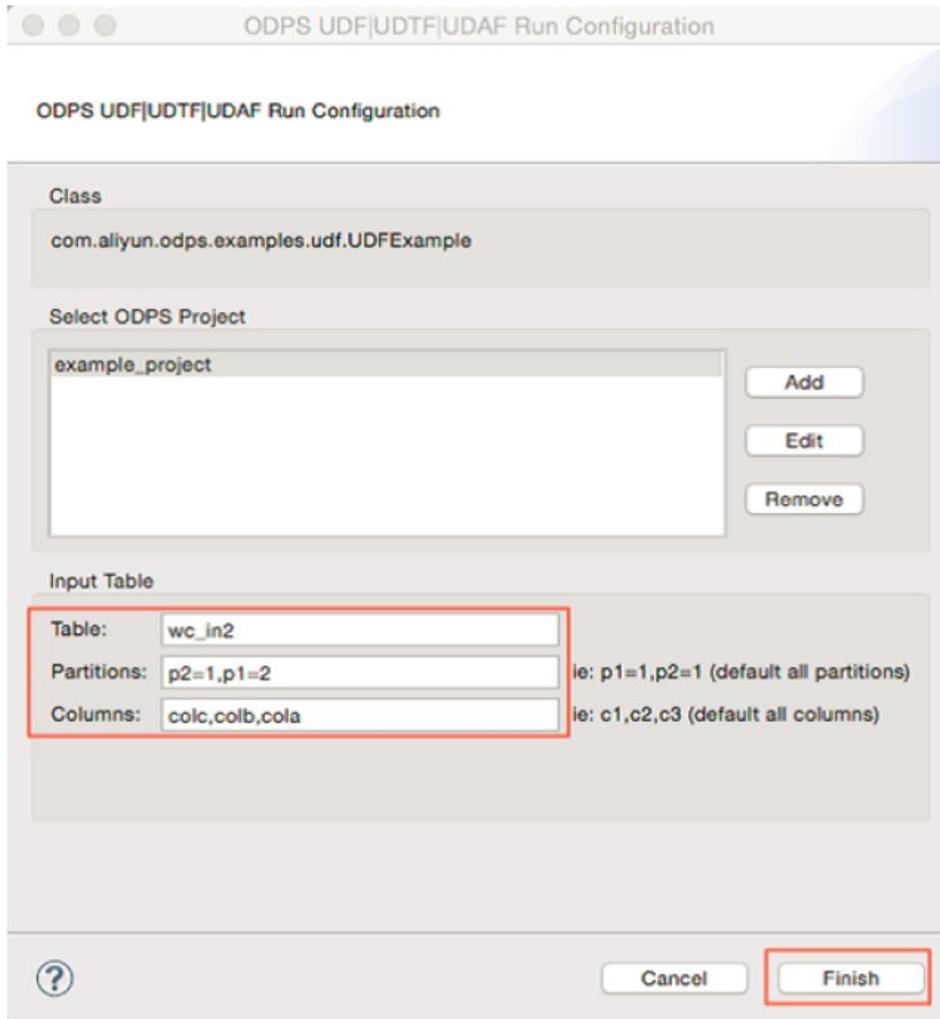
1. On the Package Explorer tab of the Eclipse development plug-in, right-click a Java file in the navigation tree, such as UDFExample.java, and choose Run As > Run UDF|UDAF|UDTF, as shown in the following figure.

Shortcut menu



2. In the ODPS UDF|UDTF|UDAF Run Configuration dialog box, configure related parameters, as shown in the following figure.

ODPS UDF|UDTF|UDAF Run Configuration dialog box



**Note** In the ODPS UDF|UDTF|UDAF Run Configuration dialog box, the Table parameter indicates the input table of the UDF. The Partitions parameter indicates the partitions from which you want to read data. Separate multiple partitions with commas (.). The Columns parameter indicates the columns that are passed as the parameters of the UDF. Separate multiple columns with commas (.).

3. Click **Finish** to run the UDF and obtain the output result.

### 23.2.4.2. Run a UDF

This topic describes how to run a user-defined function (UDF) in the Eclipse development plug-in.

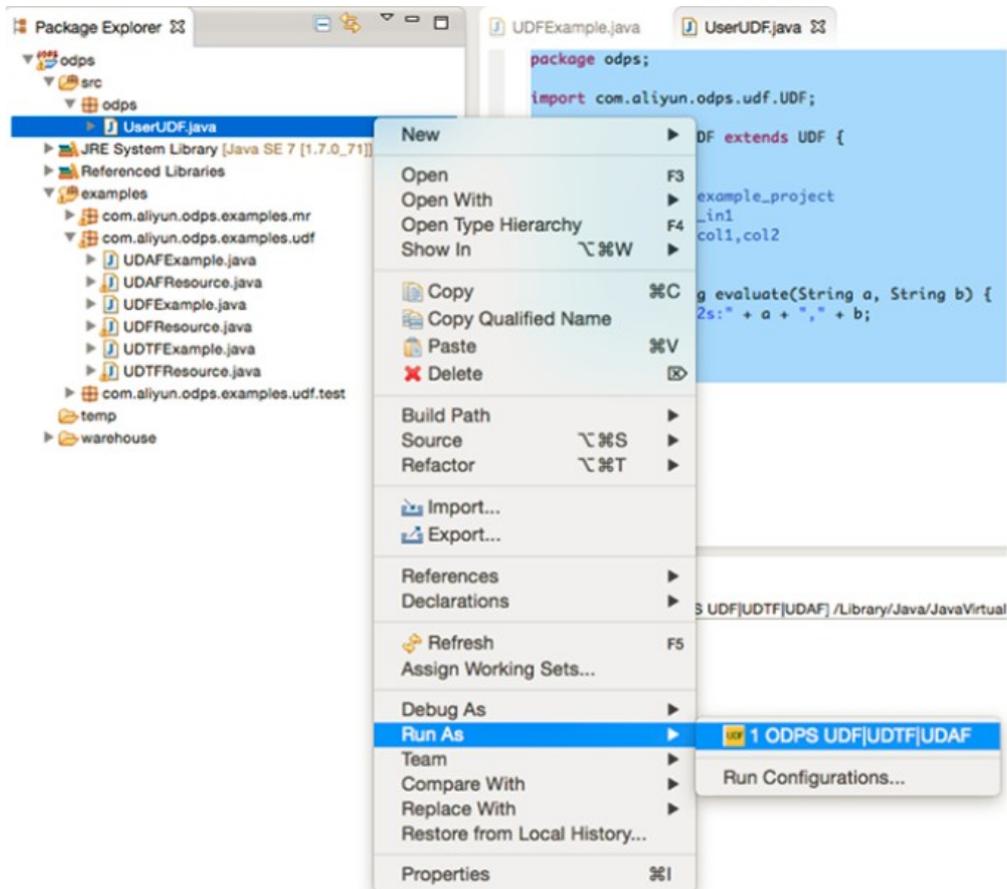
#### Procedure

1. Right-click a project and choose **New > UDF**. You can also choose **File > New > UDF** from the menu bar. Then, enter a UDF class name and click **Finish**. A Java file with the same name as the UDF class is generated in the `src` directory. Edit the content of the Java file. Sample content in the Java file:

```
package odps;
import com.aliyun.odps.udf.UDF;
public class UserUDF extends UDF {
 /**
 * project: example_project
 * table: wc_in1
 * columns: col1,col2
 *
 */
 public String evaluate(String a, String b) { return "ss2s:" + a + "," + b;
 }
}
```

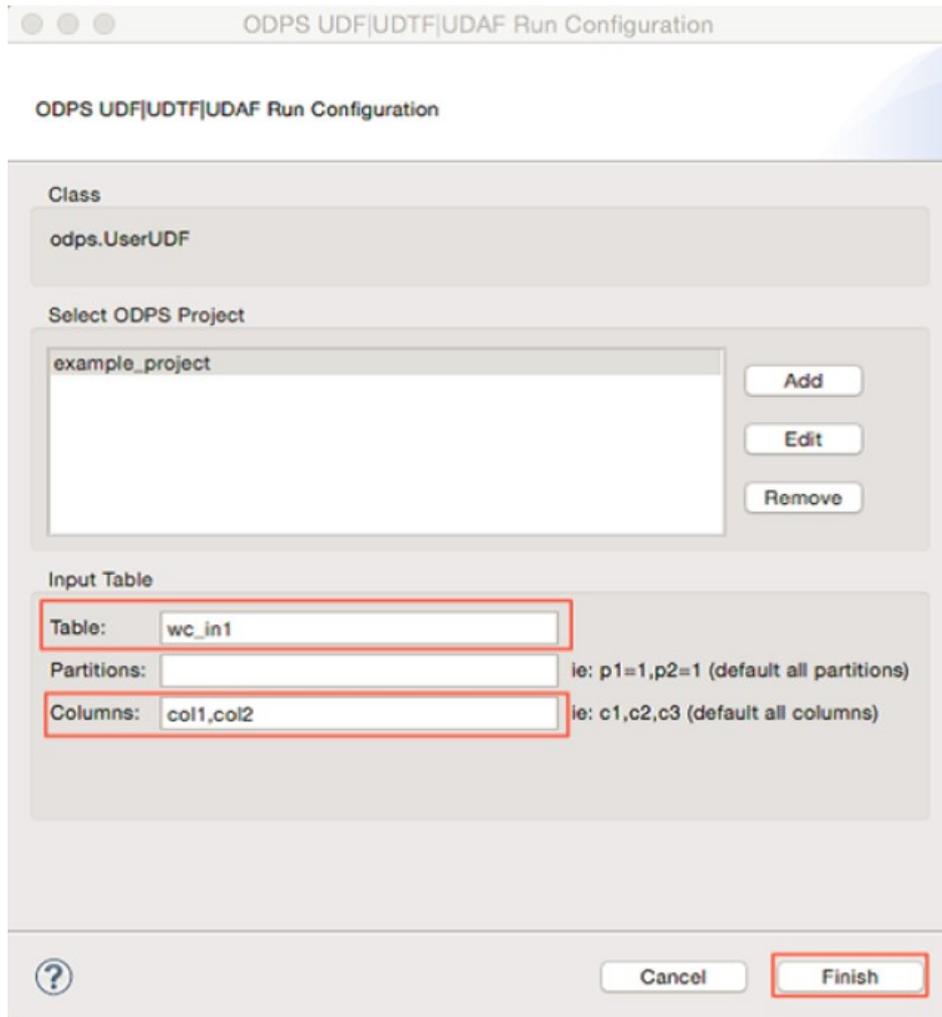
- 2. Right-click the Java file, such as the UserUDF.java file, and choose Run As > ODPS UDF|UDTF|UDAF, as shown in the following figure.

Run a UDF (1)



- 3. In the dialog box that appears, configure the parameters shown in the following figure.

Run a UDF (2)



4. Click **Finish** to obtain the result.

```
ss2s:A1,A2
ss2s:A1,A2
ss2s:A1,A2
ss2s:A1,A2
```

**Note** This example shows how to run a UDF. You can use the same method to run a user-defined table-valued function (UDTF).

## 23.2.5. Graph operation example

After you create a MaxCompute project, you can compile your own Graph program and perform the following operations to complete local debugging: This topic provides an example about how to run a Graph.

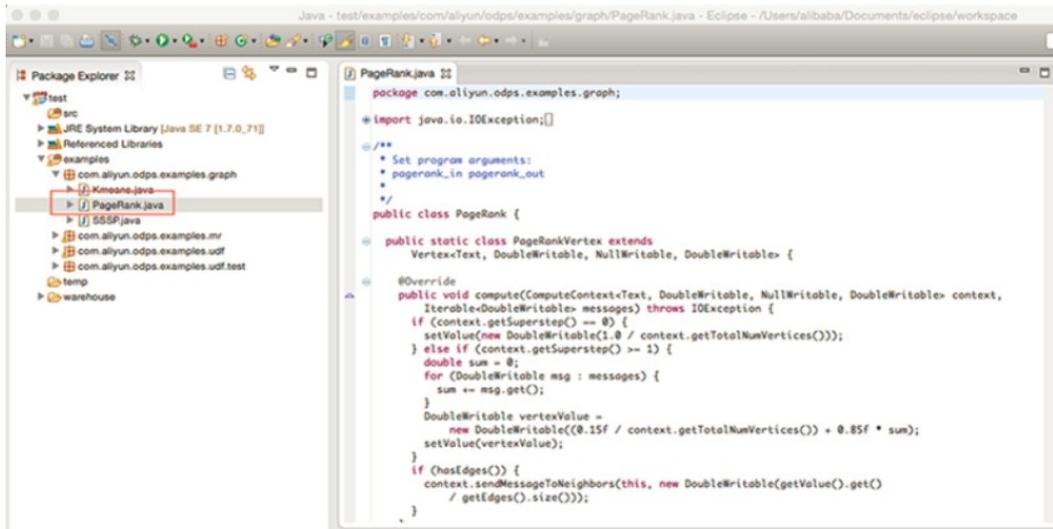
### Context

In this example, you can use the PageRank.java file provided by a plug-in to perform local debugging.

### Procedure

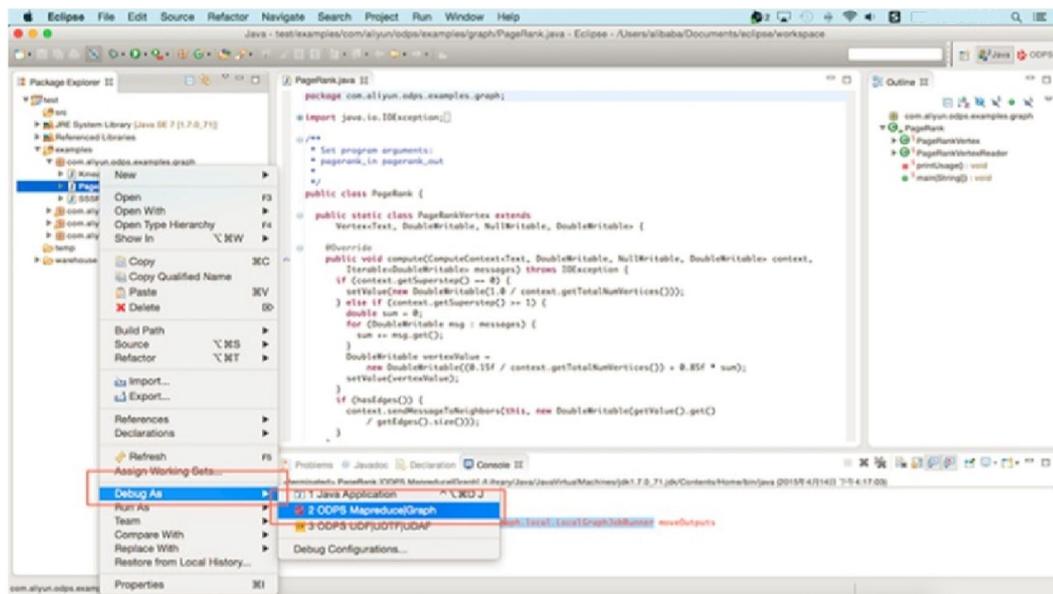
1. On the Package Explorer tab, choose `example > com.aliyun.odps.examples.graph > PageRank.java`, as shown in the following figure.

Choose PageRank.java



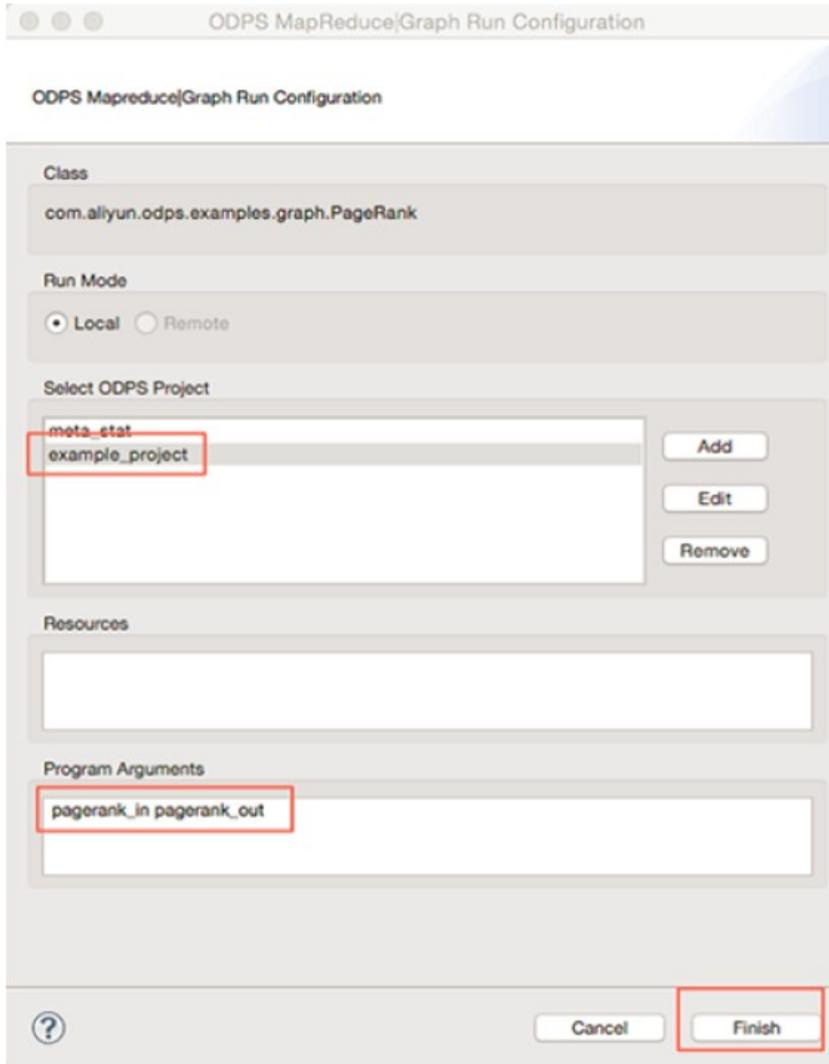
- 2. Right-click the PageRank.java file and choose **Debug As > ODPS MapReduce|Graph**, as shown in the following figure.

Choose Debug As > ODPS MapReduce|Graph



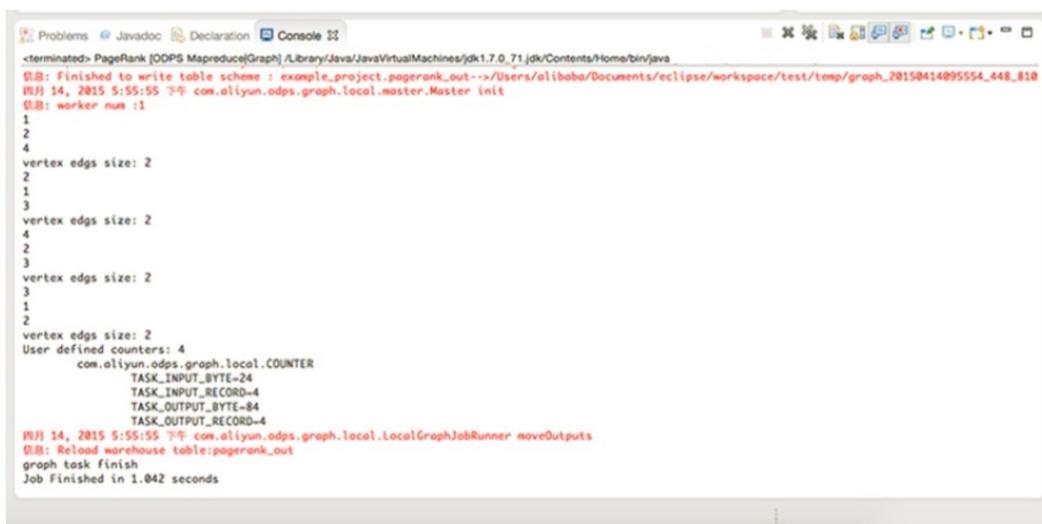
- 3. In the ODPS MapReduce/Graph Run Configuration dialog box, set the parameters to the values shown in the following figure.

ODPS MapReduce/Graph Run Configuration dialog box



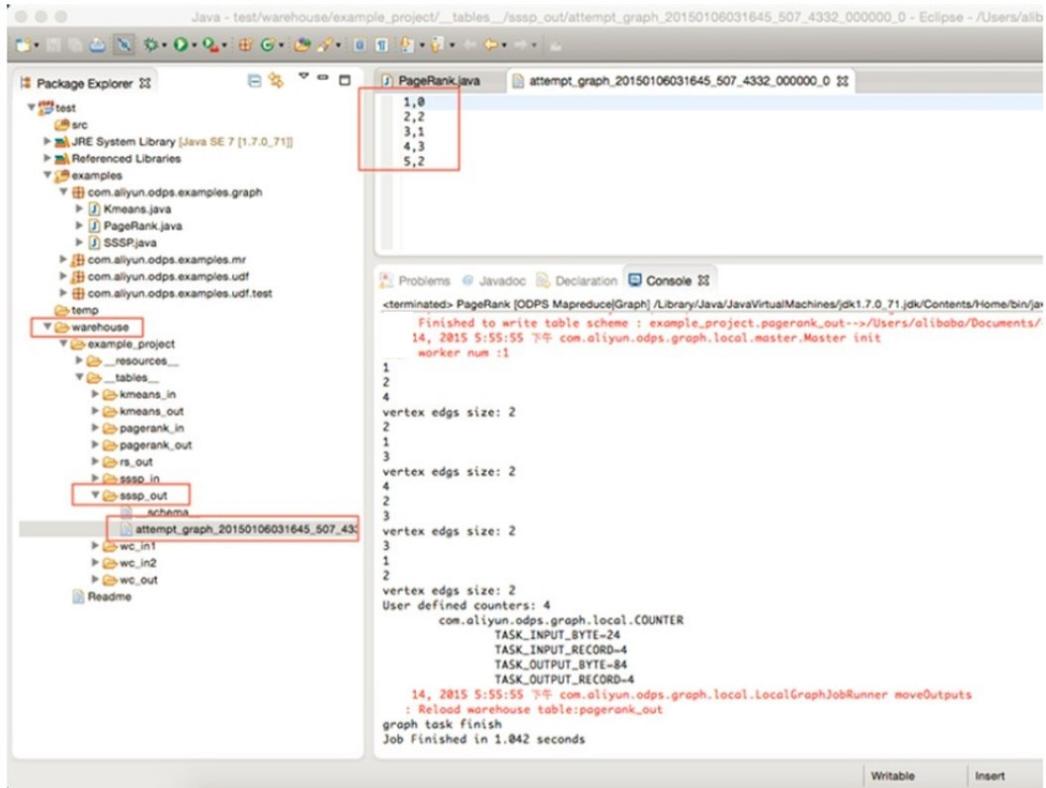
4. Click **Finish** to view the job running result, as shown in the following figure.

Running result



View the local computing result, as shown in the following figure.

Local computing result



After local debugging succeeds, you can package the program, upload the package to MaxCompute as a JAR resource, and submit a Graph job.

**Note**

- For more information about the packaging process, see [MapReduce running example](#).
- For more information about the directory structure of local results, see [MapReduce running example](#).
- For more information about how to upload JAR resources, see [Compile and run a Graph job](#).

# 24. MaxCompute feature enhancement packages

## 24.1. Content moderation

This topic describes the content moderation enhancement package.

### Purpose

The content moderation enhancement package provides an end-to-end general-purpose or industrial solution for content moderation in big data fields. This package is developed for Apsara Stack customers who have big data and content moderation requirements. In this solution, you can use MaxCompute to capture data, including structured data and unstructured data. You can also identify and review the content related to politics, pornography, and violence based on the content moderation engine.

### Description

- Content moderation

Content Moderation is a service that intelligently moderates the content of multimedia objects, such as images, videos, text, and audios in various scenarios. This service helps you effectively reduce the risk of content violations.

Common detection scenarios include intelligent pornography detection, terrorism recognition, graphic advertising recognition, logo recognition, facial recognition, QR code recognition, OCR image-text recognition, text-based anti-spam, voice-based anti-spam, and file content-based anti-spam.

Content Moderation provides the website detection feature. This feature automatically detects risky and illicit content on your website on a regular basis. The OSS violation detection feature is provided to perform pornography- and violence-related content detection on images and videos in your specified Object Storage Service (OSS) buckets. You can also directly call the Content Moderation API to submit machine identification tasks for specific scenarios.

- Types of content moderation

Content moderation is classified into the following types based on the data types and API definitions:

- Image moderation.
- Video moderation.
- Text moderation.
- Audio moderation.
- File moderation. After you submit a file moderation task, file content is automatically parsed to moderate images and text in the file.

- Unstructured data processing in MaxCompute

In addition to structured data, MaxCompute also allows you to process unstructured data. You can use external tables to process unstructured data stored in data storage services, such as OSS and Tablestore. You can also use volumes to store and process unstructured data in MaxCompute.

In content moderation scenarios, you can directly use Content Moderation to process unstructured data in OSS. In volume or table scenarios, we recommend that you perform content moderation on data in MaxCompute.

### Examples

Raw data:

```
localtest_byok>read table_iacs_demo;
```

The following result is returned:

```
+-----+
| data |
+-----+
| aaaa |
| Obama and Trump met at White House yesterday. |
| Hi, a booty call? |
| You are such a rubbish, really heartless. |
| If you have free time and want to find a part-time job, you can call me. |
| bbbb |
+-----+
```

The following content is provided after content moderation is performed:

```
localtest_byok>select IAcs_demo(data, "demo") as (code,msg,content,filteredContent,suggestion,label,results) from table_iacs_demo;
```

The following result is returned:

```
+-----+-----+-----+-----+-----+-----+-----+
| code | msg | content | filteredcontent | suggestion | label | results |
+-----+-----+-----+-----+-----+-----+-----+
| 200 | OK | aaaa | NULL | pass | normal | [{"rate":99.91,"suggestion":"pass",
"label":"normal","scene":"antispam"}] |
| 200 | OK | Obama and Trump met at White House yesterday. | Obama and *** met at White House yesterday. | block | customized | [{"rate":99.91,"suggestion":"block","details":{"contexts":{"libCode":"4386012","libName":"demo","context":"Trump"},"label":"customized"},"label":"customized","scene":"antispam"}] |
| 200 | OK | Hi, a booty call? | NULL | block | porn | [{"rate":99.91,"suggestion":"block","details":{"contexts":{"context":"Hi, a booty call?"},"label":"porn"},"label":"porn","scene":"antispam"}] |
| 200 | OK | You are such a rubbish, really heartless. | NULL | block | abuse | [{"rate":89.1,"suggestion":"block","details":{"label":"abuse"},"label":"abuse","scene":"antispam"}] |
| 200 | OK | If you have free time and want to find a part-time job, you can call me. | NULL | pass | normal | [{"rate":99.91,"suggestion":"pass","label":"normal","scene":"antispam"}] |
| 200 | OK | bbbb | NULL | pass | normal | [{"rate":99.91,"suggestion":"pass",
"label":"normal","scene":"antispam"}] |
+-----+-----+-----+-----+-----+-----+-----+
```

## 24.2. MCQA

### 24.2.1. Overview

This topic describes the architecture, key features, scenarios, and limits of MaxCompute Query Acceleration (MCQA).

#### Description

MCQA of MaxCompute accelerates the execution of small- and medium-sized query jobs and reduces the execution time from minutes to seconds. MCQA is compatible with other query features of MaxCompute.

MCQA allows you to perform ad hoc queries or business intelligence (BI) analysis by connecting mainstream BI tools or SQL clients to MaxCompute projects.

MCQA uses an independent resource pool that does not use quota groups. MCQA automatically identifies query jobs and shortens the job queue length, which improves user experience.

## Key features

MCQA provides the following key features:

- **Low-latency resource scheduling**  
Uses an efficient and low-latency resource scheduling policy and an independent resource pool.
- **Automatic identification**  
Automatically identifies the size of query jobs. MaxCompute can use MCQA to accelerate the queries or process multiple query jobs at the same time and then return the query results. This allows you to analyze query jobs of different sizes or complexities.
- **Syntax compatibility**  
Uses the same SQL syntax as MaxCompute.
- **Selection of query methods**  
Allows the MaxCompute client to use MCQA or the offline mode to run query jobs. You can also configure to forcibly use MCQA in latency-sensitive scenarios.

## Scenarios

The following table describes the scenarios for which MCQA is suitable.

Scenario	Description	Applicable scope
Ad hoc query	You can use MCQA to optimize the query performance of small- and medium-sized datasets (less than 100 GB) and perform low-latency queries on MaxCompute tables. This helps develop and analyze data.	You can specify query criteria based on your requirements, obtain query results, and adjust the query logic. In this scenario, the query latency must be within dozens of seconds. Users are data developers or data analysts who have mastered SQL skills and prefer to use the client tools that they are familiar with to analyze queries.
BI analysis	If you use MaxCompute to build an enterprise-class data warehouse, MaxCompute performs extract, transform, load (ETL) operations to process data into business-oriented and consumable aggregate data. MCQA features low latency and supports elastic concurrency and data caching. You can use MCQA with partitions and buckets in MaxCompute tables to run concurrent jobs, generate reports, analyze statistics, and analyze fixed reports at a low cost.	In most cases, the query object is aggregate data. This scenario is suitable for multidimensional queries, fixed queries, or high-frequency queries that contain small amounts of data. In this scenario, queries are latency-sensitive, and the results are returned in seconds. For example, the latency for most queries is less than 5 seconds. The time elapsed for each query varies based on the data size and query complexity.

Scenario	Description	Applicable scope
Detailed queries and analysis of large amounts of data	MCQA automatically identifies the size of query jobs. MCQA can respond to and process small-sized jobs in a timely manner, and can allocate the resources required for large-sized jobs. This helps analysts run query jobs of different sizes and complexities.	In this scenario, large amounts of historical data is queried. The size of valid data that is queried is small, and the requirement for latency is low. Users are business analysts who need to gain business insights from data, explore potential business opportunities, and validate business assumptions.

## Limits

MCQA supports only SELECT statements and does not support user-defined functions (UDFs). If you submit a statement or feature that MCQA does not support from the MaxCompute client, the MaxCompute client automatically rolls back to the common offline mode to execute the statement or feature. Other tools cannot roll back to the common offline mode to execute the submitted statement or feature that MCQA does not support.

The following table describes the limits of MCQA.

Item	Limit
Query	<ul style="list-style-type: none"> <li>If more than 1,000 MCQA query jobs are executed at the same time, the system automatically rolls back these jobs to SQL query jobs.</li> <li>If you submit an MCQA query job from the MaxCompute client, the default timeout period is 30 seconds. If you submit an MCQA query job from the ad hoc query module of DataWorks, the default timeout period is 20 seconds. If the MCQA query job times out, the system automatically rolls back the MCQA query job to an SQL query job.</li> <li>MCQA can cache only data that is stored in ALIORC tables into memory to accelerate queries.</li> <li>You cannot use MCQA to query data from external tables.</li> </ul>
Query concurrency	You can run a maximum of 120 concurrent MCQA query jobs in each MaxCompute project.

## 24.2.2. Usage notes

MaxCompute Query Acceleration (MCQA) is a built-in feature of MaxCompute. MCQA uses the native MaxCompute SQL syntax and supports MaxCompute built-in functions and permission systems. This topic describes how to use MCQA.

### Context

You can use one of the following methods to enable the MCQA feature:

- Use the MaxCompute client. For more information, see [Enable MCQA on the MaxCompute client](#).
- Use the ad hoc queries or data analytics feature of DataWorks. By default, the MCQA feature is enabled for ad hoc queries or data analytics of DataWorks. For more information, see [Enable MCQA on the Ad-Hoc Query or Manually Triggered Workflow page](#).
- Use the MaxCompute Java Database Connectivity (JDBC) driver. For more information, see [JDBC](#).

- Use a MaxCompute SDK. This method requires dependencies in a pom.xml file. For more information, see [Enable MCQA by using MaxCompute SDK for Java](#).

### Enable MCQA on the MaxCompute client

To enable the MCQA feature on the MaxCompute client, perform the following steps:

1. Download the [client](#) package of the latest version.
2. Decompress the package to a folder that contains the following subfolders:

```
bin/
conf/
lib/
plugins/
```

3. Install and configure the client. For more information, see [Configure the client](#).
4. Append the following command line to the odps\_config.ini file in the conf folder.

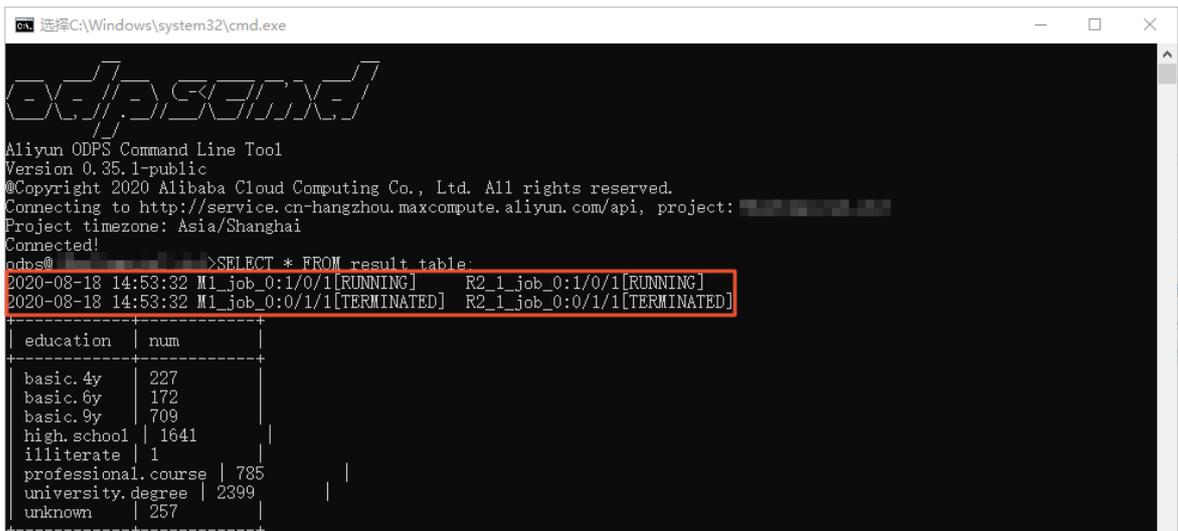
```
enable_interactive_mode=true
```

5. Run the odpscmd file in the bin directory in the Linux operating system or run the odpscmd.bat file in the bin directory in the Windows operating system. If the following information appears, the MaxCompute client runs properly.



6. Run a query job to verify that the MCQA feature is enabled.

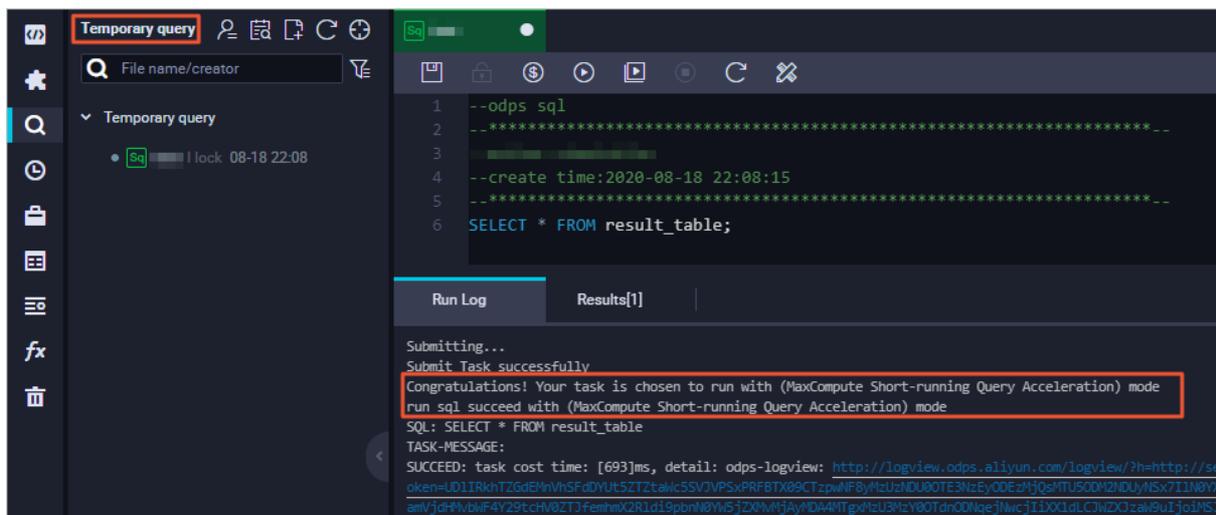
If the returned results contain the following information after you run the query job, the MCQA feature is enabled.



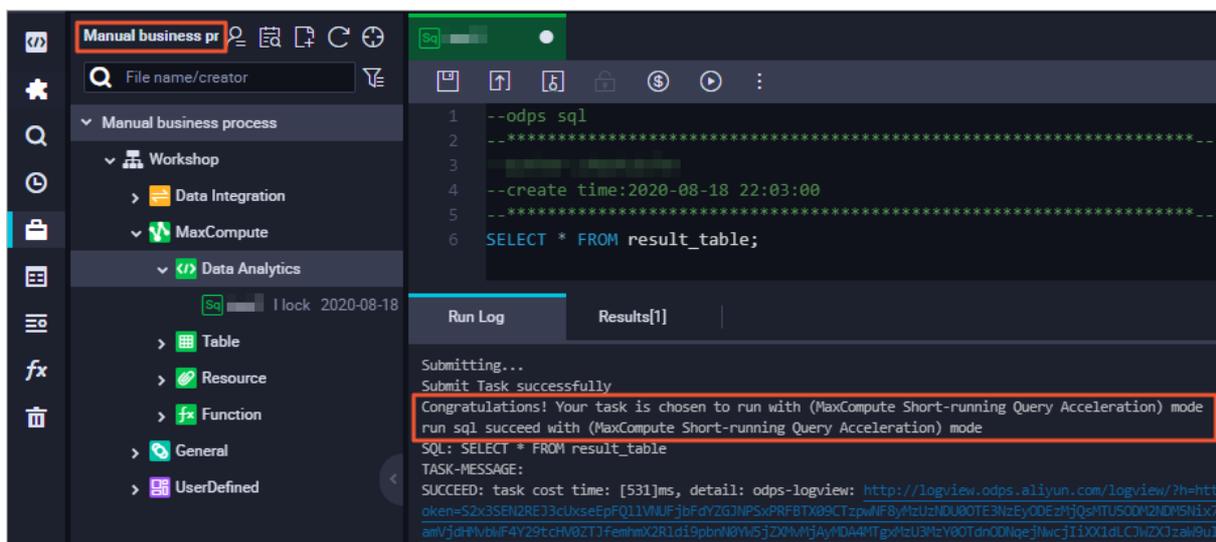
### Enable MCQA on the Ad-Hoc Query or Manually Triggered Workflow page

By default, the MCQA feature is enabled on the Ad-Hoc Query or Manually Triggered Workflow page of DataWorks. Manual operations are not required. If you want to disable the MCQA feature, submit a ticket.

Run a query job on the **Ad-Hoc Query** page. If the returned results contain the following information, the MCQA feature is enabled.



Run a query job on the **Manually Triggered Workflow** page. If the returned results contain the following information, the MCQA feature is enabled.



## JDBC

You can use the MaxCompute JDBC driver in the following scenarios:

- Use the MaxCompute JDBC driver to connect to MaxCompute. To enable the MCQA feature in this scenario, you must modify related configurations. For more information, see [Enable MCQA for the MaxCompute JDBC driver](#).
- Use the MaxCompute JDBC driver to connect to Tableau. Then, you can use Tableau to analyze data in MaxCompute in a visualized manner. To enable the MCQA feature in this scenario, you must modify related configurations. For more information, see [Enable MCQA on Tableau Server based on the MaxCompute JDBC driver](#).
- Use the MaxCompute JDBC driver to connect to SQL Workbench/J. Then, you can use SQL Workbench/J to execute SQL statements on data in MaxCompute. To enable the MCQA feature in this scenario, you must modify related configurations. For more information, see [Enable MCQA on SQL Workbench/J based on the MaxCompute JDBC driver](#).

## Enable MCQA for the MaxCompute JDBC driver

If you use the JDBC driver to connect to MaxCompute, perform the following steps to enable the MCQA feature.

1. Download the [JDBC JAR file](#) that supports the MCQA feature or download the [source code](#) that can be compiled.
2. Add the following dependency to the pom.xml file in the Maven repository:

```
<dependency>
 <groupId>com.aliyun.odps</groupId>
 <artifactId>odps-jdbc</artifactId>
 <version>3.2.0</version>
 <classifier>jar-with-dependencies</classifier>
</dependency>
```

 **Note** The version must be V3.2.0 or later.

3. Create a Java program based on the source code and configure the required information. For more information, see [ODPS JDBC](#).

The source code contains the following information:

```
String accessId = "your_access_id";
String accessKey = "your_access_key";
String conn = "jdbc:odps:http://service.odps.aliyun.com/api?project=<your_project_name>%&accessId&accessKey&charset=UTF-8&interactiveMode=true";
Statement stmt = conn.createStatement();
-- Replace your_access_id with the AccessKey ID and your_access_key with the AccessKey secret of
your Alibaba Cloud account. Replace your_project_name with the name of the project for which the
MCQA feature is enabled.
Connection conn = DriverManager.getConnection(conn, accessId, accessKey);
Statement stmt = conn.createStatement();
String tableName = "testOdpsDriverTable";
stmt.execute("drop table if exists " + tableName);
stmt.execute("create table " + tableName + " (key int, value string)");
```

To enable the MCQA feature for the MaxCompute JDBC driver, you need only to modify `String conn` or the code.

- `String conn` : Add `interactiveMode=true` .
  - Code: Add `interactiveMode=true` .
4. (Optional)Configure the following parameters in the connection string to optimize the processing logic.
    - `enableOdpsLogger` : is used to display logs. If you do not configure Simple Logging Facade for Java (SLF4J), we recommend that you set this parameter to True.
    - `fallbackForUnknownError` : The default value of this parameter is False. If this parameter is set to True, the system rolls back to the offline mode when an unknown error occurs.
    - `fallbackForResourceNotEnough` : The default value of this parameter is False. If this parameter is set to True, the system rolls back to the offline mode when resources are insufficient.
    - `fallbackForUpgrading` : The default value of this parameter is False. If this parameter is set to True, the system rolls back to the offline mode during an upgrade.
    - `fallbackForRunningTimeout` : The default value of this parameter is False. If this parameter is set to True, the system rolls back to the offline mode when an operation times out.
    - `fallbackForUnsupportedFeature` : The default value of this parameter is False. If this parameter is set to True, the system rolls back to the offline mode when the MCQA feature is not supported.
    - `fallbackForAll` : The default value of this parameter is False. If this parameter is set to True, the system rolls back to the offline mode in the preceding scenarios.

## Enable MCQA on Tableau Server based on the MaxCompute JDBC driver

Add `interactiveMode=true` to the URL of Tableau Server. We recommend that you also add `enableOdpsLogger=true` to display logs.

Sample URL of Tableau Server:

```
http://service.cn-beijing.maxcompute.aliyun.com/api?project=****_beijing&interactiveMode=true&enableOdpsLogger=true
```

To enable MCQA for some tables in a MaxCompute project, add the `table_list=table_name1, table_name2` property to the URL of Tableau Server. Then, use this property to specify the tables for which you want to enable MCQA. Separate table names with commas (.). If you specify an excessive number of tables, access to the URL of Tableau Server is time-consuming. We recommend that you specify only the required tables in the URL of Tableau Server. Sample URL:

```
http://service.cn-beijing.maxcompute.aliyun.com/api?project=****_beijing&interactiveMode=true&enableOdpsLogger=true&table_list=orders,customers
```

If a table contains a large number of partitions, we recommend that you do not use data from all partitions as the data source. You can filter the required partitions or run custom SQL queries to obtain the required data.

## Enable MCQA on SQL Workbench/J based on the MaxCompute JDBC driver

After you configure the MaxCompute JDBC driver, you can use the MCQA feature on SQL Workbench/J by modifying the JDBC URL that you specified on the profile configuration page.

Specify the JDBC URL in the following format: `jdbc:odps:<MaxCompute_endpoint>? project=<MaxCompute_project_name>&accessId=<AccessKey_ID>&accessKey=<AccessKey_Secret>&charset=UTF-8&interactiveMode=true`. Parameters:

- `maxcompute_endpoint`: the endpoint of your MaxCompute.
- `maxcompute_project_name`: the name of your MaxCompute project.
- `AccessKey ID`: the AccessKey ID that is used to access the specified project.
- `AccessKey Secret`: the AccessKey secret that corresponds to the AccessKey ID.
- `charset=UTF-8`: the character set encoding format.
- `interactiveMode`: specifies whether to enable the MCQA feature. To enable the MCQA feature, set this parameter to `true`.

## Enable MCQA by using MaxCompute SDK for Java

You must add a specified dependency to the pom.xml file in the Maven repository. Sample dependency:

```
<dependency>
 <groupId>com.aliyun.odps</groupId>
 <artifactId>odps-sdk-core</artifactId>
 <version>0.36.2-public</version>
</dependency>
```

Run commands to create a Java program. Sample commands:

```
import com.aliyun.odps.Odps;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.ResultSet;
```

```
import com.aliyun.odps.sqa.*;
import java.io.IOException;
import java.util.*;
public class SQLExecutorExample {
 public static void SimpleExample() {
 // Specify the account and project information.
 Account account = new AliyunAccount("your_access_id", "your_access_key");
 Odps odps = new Odps(account);
 odps.setDefaultProject("your_project_name");
 odps.setEndpoint("http://service.odps.aliyun.com/api");
 // Prepare to build an SQLExecutor.
 SQLExecutorBuilder builder = SQLExecutorBuilder.builder();
 SQLExecutor sqlExecutor = null;
 try {
 // run in offline mode or run in interactive mode
 if (false) {
 // Create an SQLExecutor that runs offline SQL queries by default.
 sqlExecutor = builder.odps(odps).executeMode(ExecuteMode.OFFLINE).build();
 } else {
 // Create an SQLExecutor that runs SQL queries with MCQA enabled by default.
 sqlExecutor = builder.odps(odps).executeMode(ExecuteMode.INTERACTIVE).build();
 }
 // Pass special query settings if required.
 Map<String, String> queryHint = new HashMap<>();
 queryHint.put("odps.sql.mapper.split.size", "128");
 // Submit a query job. You can pass hints.
 sqlExecutor.run("select count(1) from test_table;", queryHint);
 // List the System.out.println() statements that can be used to query common information
 .
 // UUID
 System.out.println("ExecutorId:" + sqlExecutor.getId());
 // Query the Logview URL of the current query job.
 System.out.println("Logview:" + sqlExecutor.getLogView());
 // Query the instance on which the current query job is run. In interactive mode, multiple query jobs may be run on the same instance.
 System.out.println("InstanceId:" + sqlExecutor.getInstance().getId());
 // Query the progress of the current query job. You can check the progress bar in the console.
 System.out.println("QueryStageProgress:" + sqlExecutor.getProgress());
 // Query the changelogs about the execution status for the current query job, such as rollback messages.
 System.out.println("QueryExecutionLog:" + sqlExecutor.getExecutionLog());
 // Obtain results of query jobs by calling one of the following API operations:
 if(false) {
 // Query the results of all query jobs. The API operation that you called is a synchronous operation and may occupy a thread until the query succeeds or fails.
 List<Record> records = sqlExecutor.getResult();
 printRecords(records);
 } else {
 // Query the ResultSet iterator of the query results. The API operation that you called is a synchronous operation and may occupy a thread until the query succeeds or fails.
 ResultSet resultSet = sqlExecutor.getResultSet();
 while (resultSet.hasNext()) {
 printRecord(resultSet.next());
 }
 }
 // run another query
 sqlExecutor.run("select * from test_table;", new HashMap<>());
 if(false) {
```

```

 // Query the results of all query jobs. The API operation that you called is a synchronous operation and may occupy a thread until the query succeeds or fails.
 List<Record> records = sqlExecutor.getResult();
 printRecords(records);
 } else {
 // Query the ResultSet iterator of the query results. The API operation that you called is a synchronous operation and may occupy a thread until the query succeeds or fails.
 ResultSet resultSet = sqlExecutor.getResultSet();
 while (resultSet.hasNext()) {
 printRecord(resultSet.next());
 }
 }
} catch (OdpsException e) {
 e.printStackTrace();
} catch (IOException e) {
 e.printStackTrace();
} finally {
 if (sqlExecutor != null) {
 // Close the SQLExecutor and release related resources.
 sqlExecutor.close();
 }
}
}

// SQLExecutor can be reused by pool mode
public static void ExampleWithPool() {
 // Specify the account and project information.
 Account account = new AliyunAccount("your_access_id", "your_access_key");
 Odps odps = new Odps(account);
 odps.setDefaultProject("your_project_name");
 odps.setEndpoint("http://service.odps.aliyun.com/api");
 // Run queries by using a connection pool.
 SQLExecutorPool sqlExecutorPool = null;
 SQLExecutor sqlExecutor = null;
 try {
 // Create a connection pool. Specify the connection pool size and the default execution mode.
 SQLExecutorPoolBuilder builder = SQLExecutorPoolBuilder.builder();
 builder.odps(odps)
 .initPoolSize(1) // init pool executor number
 .maxPoolSize(5) // max executors in pool
 .executeMode(ExecuteMode.INTERACTIVE); // run in interactive mode
 sqlExecutorPool = builder.build();
 // Obtain an SQLExecutor from the connection pool. If no SQLExecutor can be obtained from the connection pool, you can add an SQLExecutor. Make sure that the total number of SQLExecutors does not exceed the upper limit.
 sqlExecutor = sqlExecutorPool.getExecutor();
 // Use the SQLExecutor in the same way it is used in the preceding example.
 sqlExecutor.run("select count(1) from test_table;", new HashMap<>());
 System.out.println("InstanceId:" + sqlExecutor.getId());
 System.out.println("Logview:" + sqlExecutor.getLogView());
 List<Record> records = sqlExecutor.getResult();
 printRecords(records);
 } catch (OdpsException e) {
 e.printStackTrace();
 } catch (IOException e) {
 e.printStackTrace();
 } finally {
 sqlExecutor.close();
 }
}

```

```
 sqlExecutorPool.close();
 }
 private static void printRecord(Record record) {
 for (int k = 0; k < record.getColumnCount(); k++) {
 if (k != 0) {
 System.out.print("\t");
 }
 if (record.getColumns()[k].getType().equals(OdpsType.STRING)) {
 System.out.print(record.getString(k));
 } else if (record.getColumns()[k].getType().equals(OdpsType.BIGINT)) {
 System.out.print(record.getBigint(k));
 } else {
 System.out.print(record.get(k));
 }
 }
 }
 private static void printRecords(List<Record> records) {
 for (Record record : records) {
 printRecord(record);
 System.out.println();
 }
 }
 public static void main(String args[]) {
 SimpleExample();
 ExampleWithPool();
 }
}
```

## 24.3. VVP On MaxCompute

### 24.3.1. Overview

This topic describes VVP On MaxCompute in MaxCompute feature enhancement packages.

VVP On MaxCompute encapsulates the features of Realtime Compute for Apache Flink that is developed on the Veriverica Platform (VVP) based on MaxCompute resources. After you enable the VVP On MaxCompute feature, you can use the Cupid joint computing platform to complete the operations related to real-time computing by using the underlying storage and computing resources of MaxCompute on the VVP UI.

### 24.3.2. Benefits

This topic describes the benefits of VVP On MaxCompute.

- Seamless integration with MaxCompute
  - MaxCompute is interconnected with Realtime Compute for Apache Flink to share data. This way, Realtime Compute for Apache Flink can benefit from the scheduling and storage capabilities of MaxCompute.
  - Computing and storage resources are managed in a unified manner to ensure high elasticity. You do not need to be concerned about the issues, such as storage space expansion difficulties and time-consuming data computations that are caused by the increased data volume. The storage and retrieval capabilities of a MaxCompute cluster are automatically extended with your data volume. This enables you to focus on your business and data value.
  - Hardware integration is used to eliminate the differences in machine configuration requirements, time consumption, and peak loads of stream and batch processing. This way, the number of machines in all clusters is reduced for lower hardware costs and the model is unified for lower maintenance costs.

- Intelligent cluster deployment and O&M of Realtime Compute for Apache Flink: You can deploy Realtime Compute for Apache Flink along with MaxCompute to achieve real-time data processing and harness all the capabilities provided by MaxCompute and its related ecosystem components, such as Spark, Elasticsearch, advanced data analysis, and AI.
- Enterprise-class system security: Security isolation and control are implemented based on the multi-tenancy capability and security control of MaxCompute.
- APIs: SDK programming interfaces for Apache Flink are used to support features provided by Apache Flink and user-defined features.
- End-to-end platform development: Jobs can be created, configured, and run on MaxCompute.
- End-to-end platform O&M: Comprehensive monitoring of job running status and optimization and troubleshooting of MaxCompute are implemented. Enterprise-class services are provided, and service issues can be troubleshot and optimized.
- Excellent platform ecosystem: Real-time data computing and batch processing of data warehouses are supported on the same resource platform to integrate batch and stream processing.
- Full compatibility with enterprise-class computing engines of Apache Flink
  - Business GeminiStateBackend is provided, which brings the following benefits:
    - Uses a new data structure to accelerate ad hoc queries and reduce frequent disk I/O operations.
    - Optimizes the cache policy. If memory is sufficient, hot data is not stored in disks and cache entries do not expire after compaction.
    - Uses Java to implement GeminiStateBackend, which eliminates Java Native Interface (JNI) overheads that are caused by RocksDB.
    - Uses off-heap memory and implements an efficient memory allocator based on GeminiDB to eliminate the impact of garbage collection for Java Virtual Machines (JVMs).
    - Supports asynchronous incremental checkpoints. This ensures that only memory indexes are copied during data synchronization. Compared with RocksDB, GeminiStateBackend avoids jitters that are caused by I/O operations.
    - Supports local recovery and storage of the timer.
  - Support for various connectors: Connectors for services, such as Log Service, DataHub, Message Queue for Apache Kafka, and Tablestore, are supported. The supported connectors are continuously updated.
  - Full compatibility with Apache Flink: VVP On MaxCompute is continuously updated along with release updates of Apache Flink.

### 24.3.3. Activate VVP On MaxCompute

This topic describes how to activate VVP On MaxCompute.

**Note** You must perform the following operations as user admin, unless otherwise specified.

1. Run the following command on ODPS AG to start the Webrm service:

```
cd /home/admin/vvp/web_rm_server
sudo sh webrm.sh restart
```

After you run the command, run `netstat -nlp | grep 9088`. If the following information is displayed, the Webrm service is started.

```
$netstat -nlp|grep 9088
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp 0 0 10.5.10.203:9088 0.0.0.0:* LISTEN -
```

2. Run the following command on ODPS AG to check the running status of Docker containers.

```
ps -ef|grep vvp

$docker ps|grep vvp
4d6e2222fa99 39c894ad404b "tail -f /dev/null" 2 days ago Up 2 days
odps-service-regress.VVPUI__vvp_ui.1606281917
73e611f482b8 8d34715b6d78 "tail -f /dev/null" 2 days ago Up 2 days
odps-service-regress.VVPAppmanager__vvp_appmanager.1606281916
9b3b7ed738a6 d2aa039a3150 "tail -f /dev/null" 2 days ago Up 2 days
odps-service-regress.VVPGateway__vvp_gateway.1606281914
```

3. Log on to the vvp\_ui container based on the container ID obtained in Step 2, go to the /home/admin directory, and then run the ./start command to run the script. If you run netstat -nlp|grep 4200 in a new window on ODPS AG and the following information is displayed, the container starts.

```
$netstat -nlp|grep 4200
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp 0 0 0.0.0.0:4200 0.0.0.0:* LISTEN -
```

4. Log on to the vvp\_gateway container based on the container ID obtained in Step 2, go to the /home/admin directory, and then run the ./start command to run the script. If you run netstat -nlp|grep 8989 in a new window on ODPS AG and the following information is displayed, the container starts.

```
$netstat -nlp|grep 8989
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp 0 0 0.0.0.0:8989 0.0.0.0:* LISTEN -
```

5. Log on to the vvp\_appmanager container based on the container ID obtained in Step 2, go to the /home/admin directory, and then run the ./start command to run the script. If you run netstat -nlp|grep 9100 in a new window on ODPS AG and the following information is displayed, the container starts.

```
$netstat -nlp|grep 9100
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp 0 0 0.0.0.0:9100 0.0.0.0:* LISTEN -
```

**Note** Perform Step 3 to Step 5 in sequence. The next step can be performed only after the preceding step is complete. Random order is not allowed.

VVP On MaxCompute is activated. You can complete relevant operations on the VVP UI.

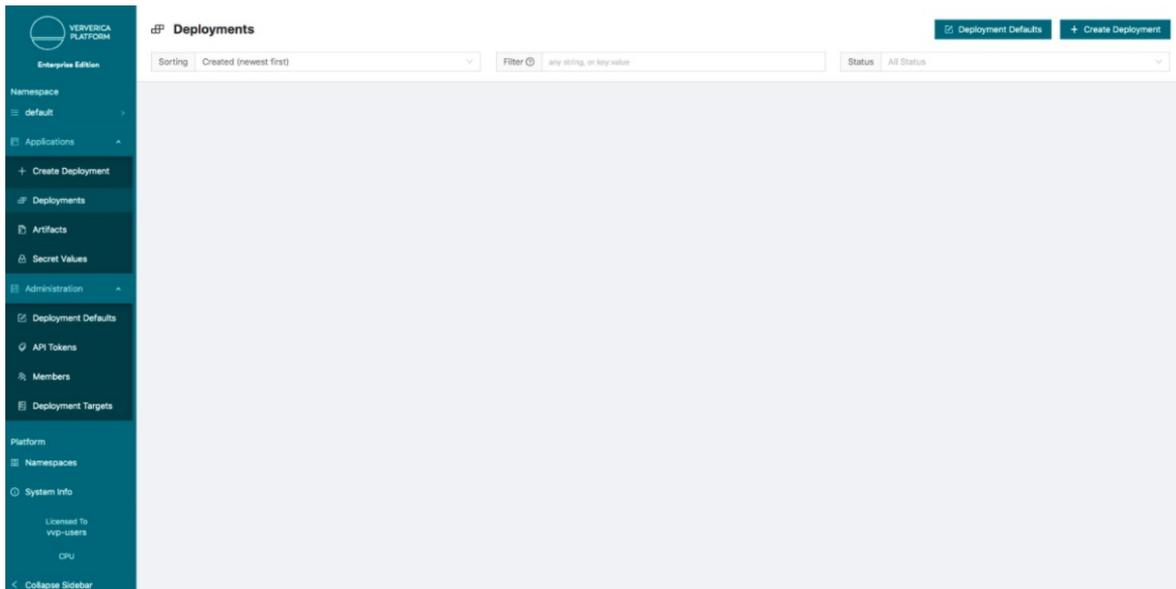
## 24.3.4. Usage notes

This topic describes how to perform operations on the VVP UI.

1. Open your browser, enter the URL of the VVP UI in the address bar, and then press Enter to log on to the VVP UI.

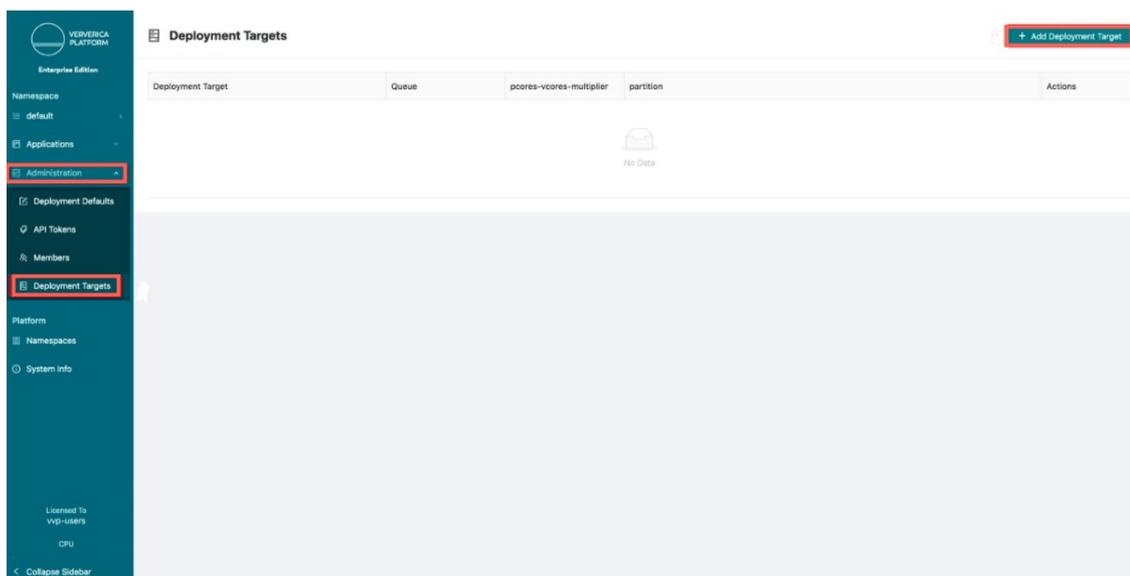
The URL of the VVP UI is in the format of xx.xx.xx.xx:8989. xx.xx.xx.xx indicates the IP address of ODPS AG and 8989 is the port number.

**Note** We recommend that you use the Google Chrome browser.



2. Create a deployment target.

- i. In the left-side navigation pane, choose **Administration > Deployment Targets**. On the Deployment Targets page, click **+ Add Deployment Target**. The **Add Deployment Target** dialog box appears.



- ii. In the Add Deployment Target dialog box, specify Deployment Target Name and Yarn Queue, and click OK.

**Note** Deployment Target Name allows you to customize the settings. Yarn Queue must be set to default.

**Add Deployment Target** X

\* Deployment Target Name  
FlinkDemo

\* Yarn Queue  
default

pcores-vcores-multiplier  
1

partition  
partition

It is not recommended to use the same Yarn Queue for Veriverica Platform and your Deployments.

Cancel OK

- iii. View the information of the deployment target that you created on the VVP UI.

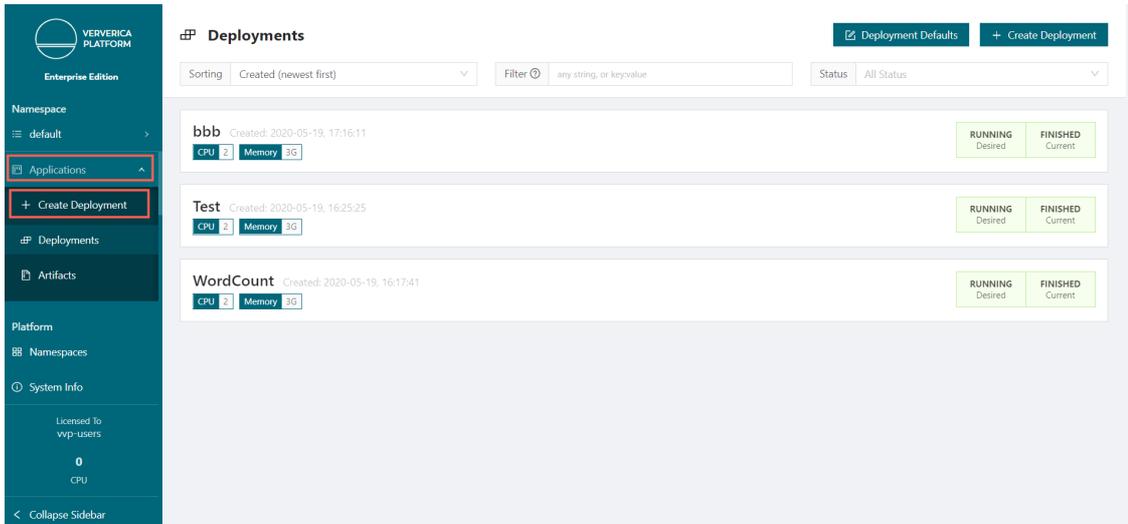
**Deployment Targets** + Add Deployment Target

Deployment Target	Queue	pcores-vcores-multiplier	partition	Actions
FlinkDemo 1ba41629-d215-4a1e-ab9c-2ead066c26cf yarn	default	1	-	Delete

- 3. Create a deployment.

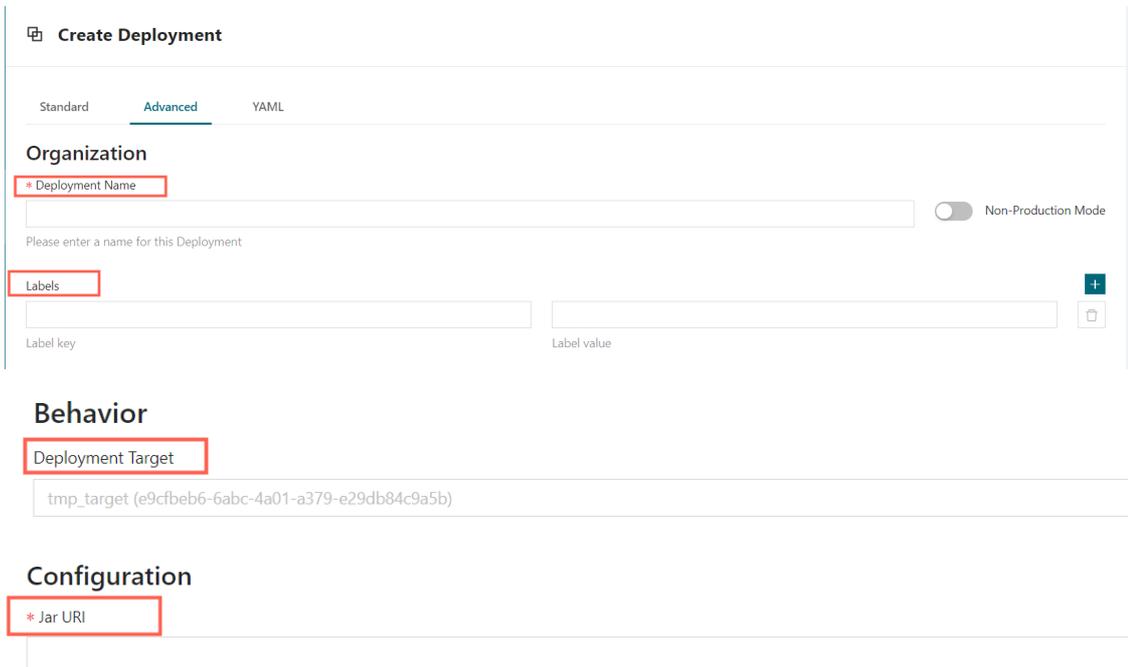
**Note** Before you create a deployment, make sure that the deployment target already exists.

- i. In the left-side navigation pane, choose **Applications > + Create Deployment**. The **Create Deployment** page appears.



- ii. Click the **Advanced** tab and configure parameters.

**Note** Deployment Name and Labels allow you to customize settings. The value of Labels is in the format of key-value pairs. Select an existing deployment target from the Deployment Target drop-down list, enter the address for downloading the JAR file in Jar URI, and then retain default values for other parameters.



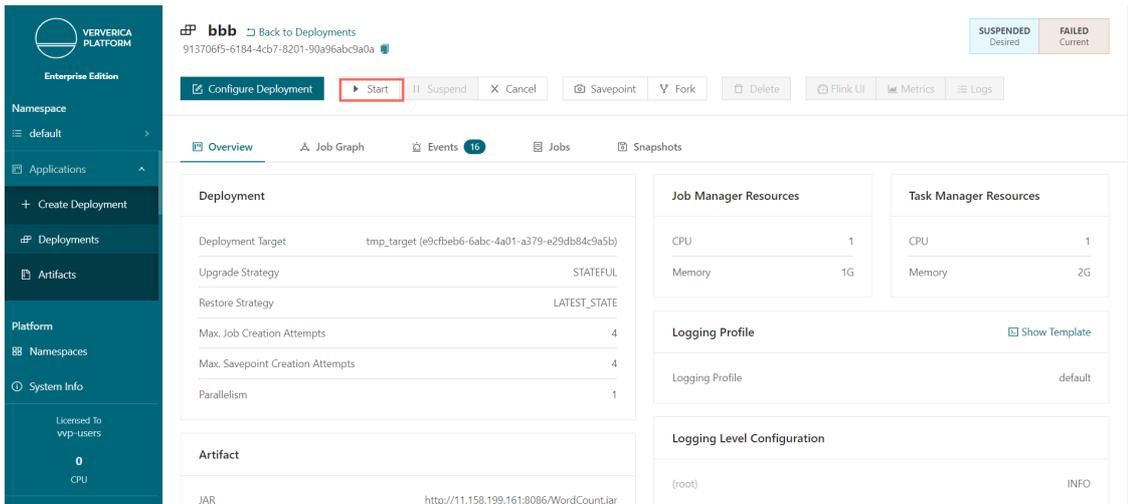
iii. Click **Create Deployment** to create the deployment.

iv. View the information of the new deployment on the VVP UI.

4. Start the deployment.

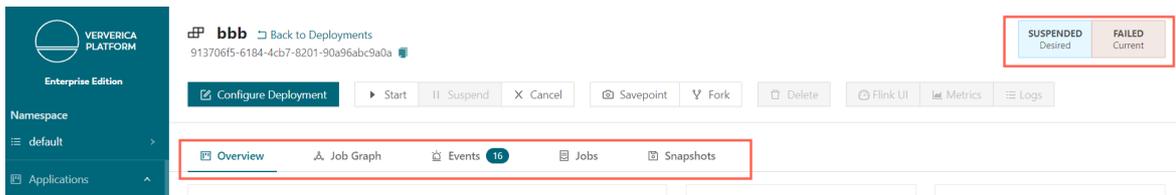
i. In the left-side navigation pane, choose **Applications > Deployments**. On the Deployments page, click the deployment that you created. The deployment details page appears.

ii. On the deployment details page, click **Start** to start the deployment.

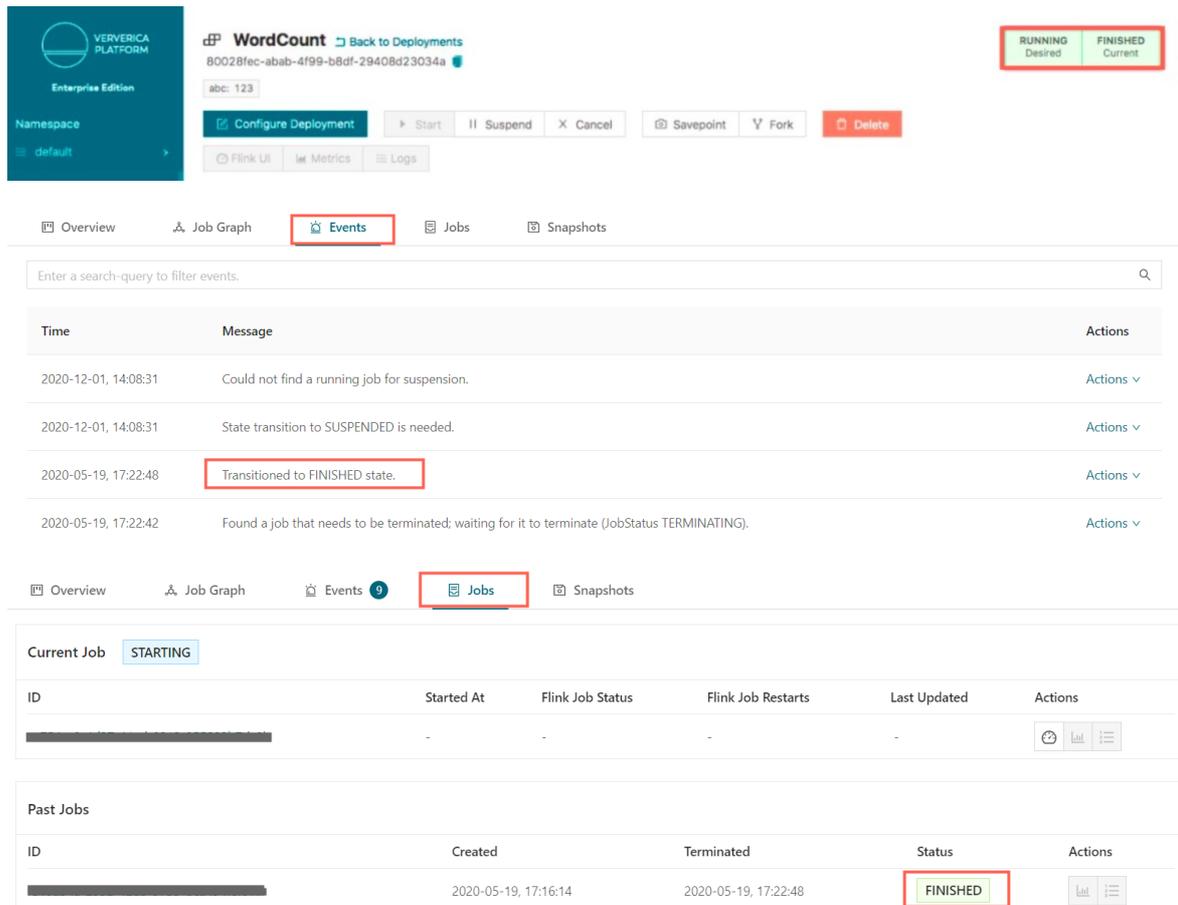


5. View the information of the deployment.

When you run the deployment, you can view the information shown on the deployment details page. The information provides the running status of the deployment.



After the deployment is complete, its status changes. You can view the deployment completion information on the **Events** and **Jobs** tabs.



If an error occurs on a running deployment, you can view further details about error information on the **Events** tab.



## 24.4. Mars

### 24.4.1. Overview

This topic describes the features of Mars, the differences between Mars and PyODPS, and the scenarios of using Mars and the PyODPS DataFrame API.

#### Scenarios

Use Mars and the PyODPS DataFrame API in the following scenarios:

- Mars
  - You often call the `to_pandas()` method of the PyODPS DataFrame API to convert a PyODPS DataFrame into a pandas DataFrame.
  - You are familiar with the pandas API, but do not want to learn the PyODPS DataFrame API.
  - You want to use indexes.

- You want to maintain the data order after you create a DataFrame.

The Mars DataFrame API provides the `iloc` method to retrieve rows and obtain data from specific rows. For example, `df.iloc[10]` is used to obtain data in the tenth row. The Mars DataFrame API also provides the `df.shift()` and `df.ffill()` methods, both of which can be used only in scenarios where the data order is maintained.

- You want to run NumPy or scikit-learn in a parallel and distributed manner, or run TensorFlow, PyTorch, and XGBoost in a distributed manner.
- You want to process data whose volume is less than 1 TB.

- PyODPS DataFrame

- You want to use MaxCompute to schedule jobs. The PyODPS DataFrame API compiles operations on DataFrames to MaxCompute SQL statements. If you want to schedule jobs by using MaxCompute, we recommend that you use the PyODPS DataFrame API.
- You want to schedule jobs in a more stable environment. The PyODPS DataFrame API compiles operations to MaxCompute SQL statements for execution. MaxCompute is stable, which means that PyODPS is stable. Mars is new and less stable. Therefore, we recommend that you use the PyODPS DataFrame API if you require high stability.
- If you want to process data whose volume is larger than 1 TB, we recommend that you use the PyODPS DataFrame API.

## Differences between Mars and PyODPS

- API

- Mars

The Mars DataFrame API is fully compatible with pandas. The Mars tensor API is compatible with NumPy. The Mars learn API is compatible with scikit-learn.

- PyODPS

PyODPS provides only the DataFrame API, which is different from the pandas API.

- Indexing

- Mars

The Mars DataFrame API supports operations based on indexes, including row indexes and column indexes. Example:

```
In [1]: import mars.dataframe as md
In [5]: import mars.tensor as mt
In [7]: df = md.DataFrame(mt.random.rand(10, 3), index=md.date_range('2020-5-1', periods=10))
In [9]: df.loc['2020-5'].execute()
Out[9]:
```

	0	1	2
2020-05-01	0.061912	0.507101	0.372242
2020-05-02	0.833663	0.818519	0.943887
2020-05-03	0.579214	0.573056	0.319786
2020-05-04	0.476143	0.245831	0.434038
2020-05-05	0.444866	0.465851	0.445263
2020-05-06	0.654311	0.972639	0.443985
2020-05-07	0.276574	0.096421	0.264799
2020-05-08	0.106188	0.921479	0.202131
2020-05-09	0.281736	0.465473	0.003585
2020-05-10	0.400000	0.451150	0.956905

- PyODPS

PyODPS does not support index-based operations.

- Data order

- Mars

After a Mars DataFrame is created, it maintains the data order. The Mars DataFrame API provides time series methods such as `shift`, and missing value handling methods such as `ffill` and `bfill`.

```
In [3]: df = md.DataFrame([[1, None], [None, 1]])
In [4]: df.execute()
Out[4]:
 0 1
0 1.0 NaN
1 NaN 1.0
In [5]: df.ffill().execute() # Fill the missing value with the value in the previous row.
Out[5]:
 0 1
0 1.0 NaN
1 1.0 1.0
```

- PyODPS

PyODPS processes and stores data by using MaxCompute, which does not maintain the data order. Therefore, PyODPS does not ensure the data order or support time series methods.

- Execution

- Mars

Mars consists of a client and a distributed execution layer. You can call the `o.create_mars_cluster` method to create a Mars cluster in MaxCompute and submit computing jobs to the Mars cluster. This process greatly reduces the costs for scheduling. Mars outperforms PyODPS in processing smaller amounts of data.

- PyODPS

PyODPS is a client and does not contain any servers. When you use the PyODPS DataFrame API, the system compiles the operations to MaxCompute SQL statements for execution. Therefore, the operations supported by the PyODPS DataFrame API depend on MaxCompute SQL. Every time you call the `execute` method, a MaxCompute job is submitted for the cluster to schedule.

## Usage notes

Mars is a unified distributed computing framework based on tensors. Mars can use parallel and distributed computing technologies to accelerate data processing for Python data science libraries such as [NumPy](#), [pandas](#) and [scikit-learn](#).

Mars provides the following common APIs:

- Mars tensor

The Mars tensor API mimics the NumPy API and can process large multidimensional arrays, which are also called tensors. The following code shows an example of how to use the Mars tensor API:

```
import mars.tensor as mt
a = mt.random.rand(10000, 50)
b = mt.random.rand(50, 5000)
a.dot(b).execute()
```

- Mars DataFrame

The Mars DataFrame API mimics the pandas API and can process and analyze a large amount of data. The following code shows an example of how to use the Mars DataFrame API:

```
import mars.dataframe as md
ratings = md.read_csv('Downloads/ml-20m/ratings.csv')
movies = md.read_csv('Downloads/ml-20m/movies.csv')
movie_rating = ratings.groupby('movieId', as_index=False).agg({'rating': 'mean'})
result = movie_rating.merge(movies[['movieId', 'title']], on='movieId')
result.sort_values(by='rating', ascending=False).execute()
```

- **Mars learn**

The Mars learn API mimics the scikit-learn API. The Mars learn API can be integrated with [TensorFlow](#), [PyTorch](#), and [XGBoost](#). The following code shows an example of how to use the Mars learn API:

```
import mars.dataframe as md
from mars.learn.neighbors import NearestNeighbors
df = md.read_csv('data.csv')
nn = NearestNeighbors(n_neighbors=10)
nn.fit(df)
neighbors = nn.kneighbors(df).fetch()
```

## Reference

- [Mars on GitHub](#)
- [Mars documentation](#)
- [Mars column](#)

## 24.4.2. Preparations

This topic describes how to prepare the Mars runtime environment.

To run Mars in MaxCompute, you must prepare the Mars runtime environment by using one of the following methods:

- **DataWorks**

- i. Create a DataWorks PyODPS 3 node, which provides features of PyODPS and Mars.

You can run the following commands in the new PyODPS 3 node to check the versions of PyODPS and Mars. Make sure that the versions meet the requirements.

```
from odps import __version__ as odps_version
from mars import __version__ as mars_version
print(odps_version)
print(mars_version)
```

`odps_version`: the version of PyODPS. Make sure that the PyODPS version is V0.9.3.1 or later. `mars_version`: the version of Mars. Make sure that the Mars version is V0.4.4 or later.

- ii. Initialize a MaxCompute entry point.

You can use the MaxCompute entry point provided by the DataWorks PyODPS 3 node.

- **Other environments**

- i. Install pip. After pip is installed, install PyODPS and Mars by running the pip install command in CLI, such as the Command Prompt in Windows. For more information about how to install pip, see [Installation](#) in the pip documentation. The following commands show an example of how to use the pip install command:

```
pip install -U pip # Optional. Make sure that pip is in the latest version.
pip install pyodps -i https://mirrors.aliyun.com/pypi/simple/ # Install the latest version of PyODPS. In the command, https://mirrors.aliyun.com/pypi/simple/ is the URL of the Python Package Index (PyPI) mirror that Alibaba Cloud provides to accelerate package download.
pip install pymars -i https://mirrors.aliyun.com/pypi/simple/ # Install the latest version of Mars.
pip install protobuf -i https://mirrors.aliyun.com/pypi/simple/ # Install the latest version of protocol buffers.
pip install pyarrow -i https://mirrors.aliyun.com/pypi/simple/ # Optional. Install the latest version of PyArrow to accelerate job execution in Mars.
```

ii. Initialize a MaxCompute entry point.

You must use your AccessKey ID and AccessKey secret to initialize the MaxCompute entry point. For more information about how to initialize a MaxCompute entry point, see [PyODPS: ODPS Python SDK and data analysis framework](#).

### 24.4.3. Usage notes

This topic describes how to perform operations on Mars clusters, read and write MaxCompute tables, and obtain the URLs of Mars UI, Logview, and Jupyter Notebook.

For more information about how to develop Mars jobs, see [Mars](#).

#### Mars cluster operations

- Create a Mars cluster

Run the following commands to create a Mars cluster. This process takes a while to complete.

```
from odps import options
options.verbose = True
If the preceding commands have been configured on the DataWorks PyODPS 3 node, you do not need to run the two commands.
client = o.create_mars_cluster(5, 4, 16, min_worker_num=3)
```

where:

- 5: the number of worker nodes in the cluster. In this example, the cluster consists of five worker nodes.
- 4: the number of CPU cores for each worker node. In this example, each worker node has four CPU cores.
- 16: the memory size of each worker node. In this example, each worker node has 16 GB of memory.

#### Note

- The memory size that you request for each worker node must be greater than 1 GB. The optimal ratio of CPU cores to the memory size is 1:4. For example, configure a worker node with 4 CPU cores and 16 GB of memory.
- You can create a maximum of 30 worker nodes. If the number of worker nodes exceeds the upper limit, the image server may be overloaded. If more than 30 workers are required, submit a ticket.

- min\_worker\_num: the minimum number of worker nodes that must be started for the system to return a client object. If this parameter is set to 3, the system returns a client object after the three worker nodes are started.

If you set `options.verbose` to True when you create a Mars cluster, the URLs of Logview, Mars UI, and Jupyter Notebook of the MaxCompute instance are displayed in the command output. You can use the Mars UI to connect to Mars clusters and query the status of clusters and jobs.

- Job

When you create a Mars cluster, the cluster creates a default session that connects to the cluster. You can call the `.execute()` method to submit a job to the cluster and run the job in the default session.

```
import mars.dataframe as md
import mars.tensor as mt
md.DataFrame(mt.random.rand(10, 3)).execute() # Call the .execute() method to submit the job to the created cluster.
```

- Stop and release a cluster

A Mars cluster is automatically released three days after it is created. If you no longer require a Mars cluster, you can call the `client.stop_server()` method to release the cluster.

```
client.stop_server()
```

## Read and write operations on MaxCompute tables

Mars can directly read and write MaxCompute tables.

- Read MaxCompute tables

Mars uses the `o.to_mars_dataframe` method to read a MaxCompute table and return a [Mars DataFrame](#).

```
In [1]: df = o.to_mars_dataframe('test_mars')
In [2]: df.head(6).execute()
Out[2]:
```

	col1	col2
0	0	0
1	0	1
2	0	2
3	1	0
4	1	1
5	1	2

- Write MaxCompute tables

Mars calls the `o.persist_mars_dataframe(df, 'table_name')` method to save a Mars DataFrame as a MaxCompute table.

```
In [3]: df = o.to_mars_dataframe('test_mars')
In [4]: df2 = df + 1
In [5]: o.persist_mars_dataframe(df2, 'test_mars_persist') # Save the Mars DataFrame as a MaxCompute table.
In [6]: o.get_table('test_mars_persist').to_df().head(6) # Call the PyODPS DataFrame API operation to query data.
```

	col1	col2
0	1	1
1	1	2
2	1	3
3	2	1
4	2	2
5	2	3

- Use the Jupyter Notebook of a Mars cluster

 **Note** The Jupyter Notebook can be used only if `with_notebook=True` is specified in `create_mars_cluster`.

When you create a Jupyter Notebook document, a session is automatically created to submit jobs to the Mars cluster. Therefore, session creation does not need to be shown in the Jupyter Notebook document.

```
import mars.dataframe as md
md.DataFrame(mt.random.rand(10, 3)).sum().execute() # Call the .execute() method in the Jupyter Notebook to submit the job to the current cluster. Therefore, session creation does not need to be shown in the Jupyter Notebook document.
```

### Note

- The Jupyter Notebook document is not automatically saved. We recommend that you manually save the Jupyter Notebook document as required.
- You can connect your Jupyter Notebook to an existing Mars cluster. For more information, see [Use an existing Mars cluster](#).

## Other operations

### ● Use an existing Mars cluster

- Recreate an existing Mars cluster based on the instance ID.

```
client = o.create_mars_cluster(instance_id=**instance-id**)
```

- To use an existing Mars cluster, create a Mars session to visit the URL of the Mars UI.

```
from mars.session import new_session
new_session(**URL of the Mars UI**).as_default() # Set the created session as the default session.
```

### ● Obtain the URL of the Mars UI

If you set `options.verbose` to `True` when you create a Mars cluster, the URL of the Mars UI is automatically displayed in the command output. You can use `client.endpoint` to obtain the URL of the Mars UI.

```
print(client.endpoint)
```

### ● Obtain the Logview URL of an instance

If you set `options.verbose` to `True` when you create a Mars cluster, the Logview URL is automatically displayed in the command output. You can also use `client.get_logview_address()` to obtain the Logview URL.

```
print(client.get_logview_address())
```

### ● Obtain the Jupyter Notebook URL

If you set `options.verbose` to `True` when you create a Mars cluster, the Jupyter Notebook URL is automatically displayed in the command output. You can also use `client.get_notebook_endpoint()` to obtain the Jupyter Notebook URL.

```
print(client.get_notebook_endpoint())
```

## 25. MaxCompute FAQ

This topic provides answers to some frequently asked questions about MaxCompute.

### How do I check MaxCompute resource usage when SQL statements are slowly executed?

Log on to the host where MaxCompute AG is deployed as user admin and perform the following steps:

1. Run the following command to sort the hosts in ascending order of the number of remaining resources on each host:

```
r tfrl|sed 's/,//g'|sort -t "|" -k2 -n
```

2. Run the following command to view resource details of each host and total cluster resources in MaxCompute:

```
r ttrl|sed 's/,//g'
```

3. Calculate the percentage of remaining resources to all resources in the cluster based on the statistics obtained in the preceding steps. Then, obtain the resource usage of MaxCompute.

### Some jobs submitted by a project are slowly executed even if remaining resources in a MaxCompute cluster are sufficient. What do I do?

A possible cause is that the resources for the quota group where the project resides are exhausted. Perform the following steps to check whether the resources are exhausted and determine whether to add resources to the quota group:

1. Log on to the host where MaxCompute AG is deployed as user admin and run the following command to check the resource usage of the quota group:

```
r quota
```

2. If resources in the quota group are exhausted, you can use Apsara Big Data Manager (ABM) to modify the settings of the quota group.

### How do I modify the settings of a quota group?

1. Log on to the host where MaxCompute AG is deployed as user admin and run the following command to create or modify quotas for the quota group.

```
sh/apsara/deploy/rpc_wrapper/rpc.sh setquota -i $QUOTAID -a $QUOTANAME -t fair -s$max_cpu_quota $max_mem_quota -m $min_cpu_quota $min_mem_quota
```

 **Note** If \$QUOTAID exists, the quotas are modified. Otherwise, quotas are created.

2. Log on to ABM to configure related settings.

### How do I perform simple operations on the metadata warehouse?

1. Log on to the host where MaxCompute AG is deployed as user admin.
2. Run the following commands:

```
/apsara/odps_tools/clt/bin/odpscmd
```

```
use meta;
```

3. Run the following command to view all tables in the metadata warehouse:

```
show tables;
```

4. Run the following command to obtain the description of a specific table:

```
desc <table>;
```

## How do I use the smart metadata warehouse?

You must install the metadata warehouse enhancement package `package+view` first. Before you install the package, make sure that you are the owner of a project that is granted the package installation permission. After you install the package, you can use the smart metadata warehouse. For more information, see [Operations for package creators](#).

### • Installation

`package+view` is a metadata warehouse enhancement package. You can run the `odpscmd --config=odps_config.ini -f init.sql` command in the *system metadata warehouse* directory to complete the installation.

#### Note

- If the system asks you to re-install the package, comment out `create package` from the first line of `odpscmd --config=odps_config.ini -f init.sql`, and then run the command again.
- `odps_config.ini` is a configuration file that contains the configuration of the account. The account you configured is also the project owner of the metadata warehouse and can access the metadata warehouse.

### • Permission assignment

Run the following command to allow you to install the `package+view` package for a project, such as the `p1` project:

```
odpscmd --config=odps_config.ini -e "allow project p1 to install package systables;"
```

Run the following command to allow you to install the `package+view` package for all projects:

```
odpscmd --config=odps_config.ini -e "allow project * to install package systables;"
```

### • User operations

You can run the following command in `odpscmd` to install the package. Before you perform this step, make sure that you are the owner of the project that is granted the package installation permission.

```
install package meta.systables;
```

After the installation is complete, you can run the following command to query the description of views in the package:

```
desc package meta.systables;
```

 **Note** After the preceding operations are complete, you can use the smart metadata warehouse.

### • Views

To learn more about the definition of the table schema, see the related content in the view description, which is displayed after you run the following command:

```
desc viewname
```

View name	Content
allowed_package_installers	Information about the project that is granted the package installation permission
column_label_grants	Column label authorization information
column_labels	Column label information of a table
columns	Table schema information
installed_packages	Information about the package installed for the project
object_privileges	Authorization information of tables, UDFs, and resources
package_resources	Object information contained in the package
partitions	Partition information of a partitioned table
policies	Policies that define user or role permissions
resources	Resource information
roles	Role information
table_label_grants	Label authorization information of a table
table_labels	Label information of a table
tables	Table or view information
tasks	Job execution records
tunnels	Data upload and download records
udf_resources	Information of resources used in UDFs
udfs	Information of UDFs
user_roles	Information of associations between users and roles
users	User information

- **Notes:**

- By default, the package+view package only allows you to query metadata in the last 180 days.
- If you do not run a job on a day, the metadata warehouse does not store the metadata of the day.
- We recommend that you specify a query range to prevent a full data scan in the last 180 days.
- The metadata of the previous day can be queried only after the metadata is generated by the metadata warehouse in the early morning of the next day. In the current configuration, metadata can be queried immediately after it is generated by the metadata warehouse.
- The metadata warehouse does not provide metadata on the day when a project is created. The purpose is to obtain only the metadata of the projects with unique names.

## How do I grant Java sandbox permissions?

1. Log on to AdminConsole and choose **MaxCompute Configuration > Project Management**. Select the

project to which you want to grant Java sandbox permissions and double-click the project to open the property dialog box.

2. In the **ODPS Sandbox Setting** section, enter the method or class that you want to use in Sandbox Java Permissions.

**Note** Make sure that the content you entered is in a valid format. The following example is for reference only. Enter each item in a single line and end it with a semicolon (;).

```
permission java.lang.RuntimePermission "readSystemProperty"; permission java.lang.RuntimePermiss
ion "modifyThreadGroup"; permission java.security.AllPermission;
```

3. Click **Finish Modification**.

## What do I do if disk space is insufficient?

In most cases, you can delete scripts to release the disk space. The most possible cause is that the root directory of MaxCompute AG or the /apsara directory occupies too much disk space. Therefore, you must delete scripts in the two directories.

## How do I add a MaxCompute host to a blacklist?

1. Log on to Apsara AG as user admin. Run the following command to enable the Fuxi blacklisting function:

```
r sgf fuximaster "{\"fuxi_Enable_BadNodeManager\":false}"
```

2. Run the following command to view the Fuxi blacklist:

```
/apsara/deploy/rpc_wrapper/rpc.shblacklist cluster get
```

3. Run the following command to add the host to the Fuxi blacklist:

```
/apsara/deploy/rpc_wrapper/rpc.shblacklist cluster add $hostname
```

4. Run the following command to check that the host has been added to the Fuxi blacklist:

```
/apsara/deploy/rpc_wrapper/rpc.shblacklist cluster get
```

## How do I export data from MaxCompute?

You can use a Tunnel command to export data. You can also configure synchronization tasks in DataWorks to export data from MaxCompute to other destinations.

## How do I view the MaxCompute version?

Run the following commands to obtain the version of MaxCompute:

```
cat /apsara/odps_info/version|grep odps
```

```
cat /apsara/version
```

## How do I restart services in MaxCompute?

1. Run the following commands to save the configurations of resident services of MaxCompute to a file. This configuration file is required when you restart MaxCompute services.

```
ssh odpsAG
cd /home/admin/
If you do not use a service, you can ignore the command that is not required.
You can run the r al command to view resident services.
r plan Odps/CGServiceControllerx > CGServiceControllerx
r plan sys/sqlonline-OTS >sqlonline-OTS
r plan Odps/MessengerService >MessengerService
r plan Odps/OdpsService >OdpsService
r plan Odps/HiveServerx >HiveServerx
r plan Odps/XStreamService >XStreamService
r plan Odps/QuotaService > QuotaService
r plan Odps/ReplicationService >ReplicationService
```

2. Run the following commands to stop services in MaxCompute:

```
r sstop Odps/CGServiceControllerx
r sstop sys/sqlonline-OTS
r sstop Odps/MessengerService
r sstop Odps/OdpsService
r sstop Odps/HiveServerx
r sstop Odps/XStreamService
r sstop Odps/QuotaService
r sstop Odps/ReplicationService
```

3. Run the following commands to start services in MaxCompute:

```
ssh odpsAG
cd /home/admin/
r start CGServiceControllerx
r start sqlonline-OTS
r start MessengerService.txt
r start OdpsService.txt
r start HiveServerx.txt
r start XStreamService.txt
r start QuotaService.txt
r start ReplicationService.txt
```

## How do I power off MaxCompute and then power it on?

1. Run the following commands to save the configurations of resident services of MaxCompute to a file. This configuration file is required when you restart MaxCompute services.

```
ssh odpsAG
cd /home/admin/
If you do not use a service, you can ignore the command that is not required.
You can run the r al command to view resident services.
r plan Odps/CGServiceControllerx > CGServiceControllerx
r plan sys/sqlonline-OTS >sqlonline-OTS
r plan Odps/MessengerService >MessengerService
r plan Odps/OdpsService >OdpsService
r plan Odps/HiveServerx >HiveServerx
r plan Odps/XStreamService >XStreamService
r plan Odps/QuotaService > QuotaService
r plan Odps/ReplicationService >ReplicationService
```

2. Run the following commands to stop services in MaxCompute:

```
r sstop Odps/CGServiceControllerx
r sstop sys/sqlonline-OTS
r sstop Odps/MessengerServicex
r sstop Odps/OdpsServicex
r sstop Odps/HiveServerx
r sstop Odps/XStreamServicex
r sstop Odps/QuotaServicex
r sstop Odps/ReplicationServicex
```

3. Run the following command to shut down the Apsara distributed operating system:

```
/home/admin/dayu/bin/allapsara stop
```

4. Run the following command to shut down compute nodes gracefully:

```
Shutdown
```

5. Start compute nodes.

6. Run the following command to start the Apsara distributed operating system:

```
/home/admin/dayu/bin/allapsara start
```

7. Run the following commands to start services in MaxCompute:

```
ssh odpsAG
cd /home/admin/
r start CGServiceControllerx
r start sqlonline-OTS
r start MessengerServicex.txt
r start OdpsServicex.txt
r start HiveServerx.txt
r start XStreamServicex.txt
r start QuotaServicex.txt
r start ReplicationServicex.txt
```

## How do I reduce the heavy load on a host?

1. Log on to the host with a heavy load and run the `top` command to check whether task processes occupy a large number of resources.
2. If the check result shows that task processes occupy a large number of resources, terminate the tasks after these tasks are complete or after communication with users.

# 26. Open source features of MaxCompute

This topic describes open source features of MaxCompute.

## SDK

MaxCompute provides APIs for SDK for Java and SDK for Python to create, view, and delete MaxCompute tables. You can use SDKs to edit code and perform required operations on MaxCompute.

Technical support: View the official documentation or submit a ticket.

## MaxCompute RODPS

MaxCompute RODPS is an R plug-in for MaxCompute. For more information, see [aliyun-odps-r-plugin](#) on GitHub.

Technical support: Leave a message or create an issue in [aliyun-odps-r-plugin](#) on GitHub.

## MaxCompute JDBC

MaxCompute JDBC is an official JDBC driver provided by MaxCompute. It provides a set of interfaces to run SQL tasks for Java programs. The project is hosted in [aliyun-odps-jdbc](#) on GitHub.

Technical support: Leave a message or create an issue in [aliyun-odps-jdbc](#) on GitHub.

## Data Collector

Data Collector is a collection of the major open source data collection tools of MaxCompute, such as the Flume plug-in, OGG plug-in, Sqoop, Kettle plug-in, and Hive Data Transfer UDTF.

Flume and OGG plug-ins are implemented based on the DataHub SDK, whereas Sqoop, Kettle plug-in, and Hive Data Transfer UDTF are implemented based on the Tunnel SDK. DataHub is a real-time data transfer channel, and Tunnel is a batch data transfer channel. The Flume and OGG plug-ins are used to transfer data in real time. Sqoop, Kettle plug-in, and Hive Data Transfer UDTF are used to transfer data in offline mode.

For information about the source code, see [aliyun-maxcompute-data-collectors](#) on GitHub. For more information about these tools, see [wiki](#) on GitHub.

Technical support: Leave a message or create an issue in [aliyun-maxcompute-data-collectors](#) on GitHub.