

Alibaba Cloud

Apsara Stack Enterprise

MaxCompute
User Guide

Product Version: V3.12.0

Document Version: 20220928

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 .
<code>Courier font</code>	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?	36
2.Usage notes	38
3.Preparations	40
3.1. Log on to the ASCM console	40
3.2. Create an Apsara Stack tenant account	40
3.3. Create a project	41
4.Quick start	44
4.1. Overview	44
4.2. Configure the MaxCompute client	45
4.3. Add and delete users	46
4.4. Grant and view permissions	47
4.4.1. Overview	47
4.4.2. ACL authorization	47
4.4.3. Policy authorization	48
4.4.4. View permissions	49
4.5. Create and authorize a role	50
4.6. Create or delete a table	51
4.6.1. Create a table	51
4.6.2. Obtain table information	51
4.6.3. Delete a table	52
4.7. Import or export data	52
4.8. Run SQL	52
4.8.1. Overview	52
4.8.2. SELECT statement	52
4.8.3. INSERT statement	53
4.8.4. JOIN statement	54

4.8.5. Other limits	54
4.9. Compile and use UDFs	54
4.9.1. Overview	54
4.9.2. UDF example	54
4.9.3. UDAF example	55
4.9.4. UDTF example	56
4.10. Compile and run a MapReduce job	57
4.11. Compile and run a Graph job	57
4.12. View job running information	59
5.Basic concepts and common commands	64
5.1. Terms	64
5.2. Common commands	70
5.2.1. Introduction	70
5.2.2. Project operations	70
5.2.3. Table operations	72
5.2.4. Instance operations	76
5.2.5. Resource operations	80
5.2.6. Function operations	82
5.2.7. Time zone configuration operations	84
5.2.8. Tunnel operations	85
5.2.9. Other operations	92
6.MaxCompute SQL	98
6.1. Overview	98
6.1.1. Scenarios	98
6.1.2. Reserved words	98
6.1.3. Partitioned table	98
6.1.4. Type conversion	99
6.1.4.1. Explicit type conversion	99

6.1.4.2. Implicit type conversion and its scope	100
6.1.4.3. SQL built-in functions	103
6.1.4.4. CASE WHEN	104
6.1.4.5. Partition column	104
6.1.4.6. UNION ALL	104
6.1.4.7. Conversion between string and datetime types	104
6.2. Operators	105
6.2.1. Relational operators	105
6.2.2. Arithmetic operators	106
6.2.3. Bitwise operators	107
6.2.4. Logical operators	108
6.3. DDL statements	109
6.3.1. Table operations	109
6.3.1.1. Create a table (CREATE TABLE)	109
6.3.1.2. Delete a table	112
6.3.1.3. Rename a table	112
6.3.1.4. Modify the comment of a table	113
6.3.1.5. Modify the lifecycle of a table	113
6.3.1.6. Disable or restore the lifecycle feature	114
6.3.1.7. Modify the LastDataModifiedTime value of a table	115
6.3.1.8. Clear data from a non-partitioned table	115
6.3.1.9. Archive table data	116
6.3.1.10. Forcibly delete data from a table (partition)	117
6.3.2. View-based operation	117
6.3.2.1. Create a view	117
6.3.2.2. Delete a view	118
6.3.2.3. Rename a view	118
6.3.3. Column and partition operations	119

6.3.3.1. Add a partition (ADD PARTITION)	119
6.3.3.2. Delete a partition (DROP PARTITION)	120
6.3.3.3. Add a column	120
6.3.3.4. Change a column name	120
6.3.3.5. Modify the comment of a column or partition	121
6.3.3.6. Modify the LastDataModifiedTime value of a partiti...	121
6.3.3.7. Modify partition values	121
6.3.3.8. Merge partitions	122
6.4. DML statements	124
6.4.1. INSERT statement	124
6.4.1.1. Update the data of a table	124
6.4.1.2. Output data to multiple objects	125
6.4.1.3. Output data to a dynamic partition	126
6.4.2. SELECT statement	127
6.4.2.1. SELECT	127
6.4.2.2. Subquery	131
6.4.3. UNION statements	132
6.4.3.1. UNION ALL	132
6.4.4. JOIN statement	133
6.4.4.1. JOIN	133
6.4.4.2. MAPJOIN HINT	135
6.4.5. EXPLAIN statement	136
6.4.6. GROUPING SETS	138
6.4.6.1. Overview	138
6.4.6.2. Example	139
6.4.6.3. CUBE and ROLLUP	140
6.4.6.4. GROUPING and GROUPING_ID	140
6.4.7. IF statement	141

6.5. SELECT TRANSFORM	142
6.5.1. Overview	142
6.5.2. SELECT TRANSFORM examples	144
6.5.2.1. Call Shell scripts	144
6.5.2.2. Call Python scripts	144
6.5.2.3. Call Java scripts	145
6.5.2.4. Call scripts of other languages	147
6.5.2.5. Call scripts in series	147
6.5.3. Performance advantages	147
6.6. UNION, INTERSECT, and EXCEPT	148
6.7. Built-in functions	151
6.7.1. Mathematical functions	151
6.7.1.1. ABS	151
6.7.1.2. ACOS	152
6.7.1.3. ASIN	152
6.7.1.4. ATAN	153
6.7.1.5. CEIL	153
6.7.1.6. CONV	154
6.7.1.7. COS	154
6.7.1.8. COSH	154
6.7.1.9. COT	155
6.7.1.10. EXP	155
6.7.1.11. FLOOR	155
6.7.1.12. LN	156
6.7.1.13. LOG	156
6.7.1.14. POW	156
6.7.1.15. RAND	157
6.7.1.16. ROUND	157

6.7.1.17. SIN	158
6.7.1.18. SINH	158
6.7.1.19. SQRT	158
6.7.1.20. TAN	159
6.7.1.21. TANH	159
6.7.1.22. TRUNC	159
6.7.1.23. Additional mathematical functions	160
6.7.1.24. LOG2	160
6.7.1.25. LOG10	161
6.7.1.26. BIN	161
6.7.1.27. HEX	161
6.7.1.28. UNHEX	162
6.7.1.29. RADIANS	162
6.7.1.30. DEGREES	163
6.7.1.31. SIGN	163
6.7.1.32. E	163
6.7.1.33. PI	164
6.7.1.34. FACTORIAL	164
6.7.1.35. CBRT	164
6.7.1.36. SHIFTLEFT	165
6.7.1.37. SHIFTRIGHT	165
6.7.1.38. SHIFTRIGHTUNSIGNED	166
6.7.2. String processing functions	166
6.7.2.1. CHAR_MATCHCOUNT	166
6.7.2.2. CHR	166
6.7.2.3. CONCAT	167
6.7.2.4. INSTR	167
6.7.2.5. IS_ENCODING	168

6.7.2.6. KEYVALUE	168
6.7.2.7. LENGTH	169
6.7.2.8. LENGTHB	170
6.7.2.9. MD5	170
6.7.2.10. PARSE_URL	170
6.7.2.11. REGEXP_EXTRACT	171
6.7.2.12. REGEXP_INSTR	172
6.7.2.13. REGEXP_SUBSTR	172
6.7.2.14. REGEXP_COUNT	173
6.7.2.15. SPLIT_PART	173
6.7.2.16. REGEXP_REPLACE	174
6.7.2.17. SUBSTR	175
6.7.2.18. TOLOWER	176
6.7.2.19. TOUPPER	176
6.7.2.20. TO_CHAR	176
6.7.2.21. TRIM	177
6.7.2.22. LTRIM	177
6.7.2.23. RTRIM	178
6.7.2.24. REVERSE	178
6.7.2.25. SPACE	178
6.7.2.26. REPEAT	179
6.7.2.27. ASCII	179
6.7.2.28. URL_ENCODE	180
6.7.2.29. URL_DECODE	180
6.7.2.30. Additional string processing functions	181
6.7.2.31. CONCAT_WS	181
6.7.2.32. LPAD	181
6.7.2.33. RPAD	182

6.7.2.34. REPLACE	182
6.7.2.35. SOUNDEX	183
6.7.2.36. SUBSTRING_INDEX	183
6.7.2.37. TRANSLATE	183
6.7.2.38. JSON_TUPLE	184
6.7.3. Date processing functions	186
6.7.3.1. DATEADD	186
6.7.3.2. DATEDIFF	187
6.7.3.3. DATEPART	188
6.7.3.4. DATETRUNC	188
6.7.3.5. GETDATE	189
6.7.3.6. ISDATE	189
6.7.3.7. LASTDAY	189
6.7.3.8. TO_DATE	190
6.7.3.9. TO_CHAR	190
6.7.3.10. UNIX_TIMESTAMP	191
6.7.3.11. FROM_UNIXTIME	191
6.7.3.12. WEEKDAY	192
6.7.3.13. WEEKOFYEAR	192
6.7.3.14. Additional date functions	193
6.7.3.15. YEAR	193
6.7.3.16. QUARTER	194
6.7.3.17. MONTH	194
6.7.3.18. DAY	194
6.7.3.19. DAYOFMONTH	195
6.7.3.20. HOUR	195
6.7.3.21. MINUTE	195
6.7.3.22. SECOND	196

6.7.3.23. FROM_UTC_TIMESTAMP	196
6.7.3.24. CURRENT_TIMESTAMP	197
6.7.3.25. ADD_MONTHS	197
6.7.3.26. LAST_DAY	197
6.7.3.27. NEXT_DAY	198
6.7.3.28. MONTHS_BETWEEN	198
6.7.3.29. EXTRACT	199
6.7.4. Window functions	199
6.7.4.1. Overview	200
6.7.4.2. COUNT	200
6.7.4.3. AVG	201
6.7.4.4. MAX	202
6.7.4.5. MIN	202
6.7.4.6. MEDIAN	203
6.7.4.7. STDDEV	203
6.7.4.8. STDDEV_SAMP	203
6.7.4.9. SUM	204
6.7.4.10. DENSE_RANK	205
6.7.4.11. RANK	206
6.7.4.12. LAG	208
6.7.4.13. LEAD	208
6.7.4.14. PERCENT_RANK	209
6.7.4.15. ROW_NUMBER	209
6.7.4.16. CLUSTER_SAMPLE	211
6.7.4.17. NTILE	212
6.7.4.18. NTH_VALUE	213
6.7.4.19. CUME_DIST	214
6.7.4.20. FIRST_VALUE	215

6.7.4.21. LAST_VALUE	217
6.7.5. Aggregate functions	219
6.7.5.1. Overview	219
6.7.5.2. COUNT	219
6.7.5.3. AVG	220
6.7.5.4. MAX	221
6.7.5.5. MIN	222
6.7.5.6. MEDIAN	222
6.7.5.7. STDDEV	222
6.7.5.8. STDDEV_SAMP	223
6.7.5.9. SUM	223
6.7.5.10. WM_CONCAT	224
6.7.5.11. PERCENTILE	224
6.7.5.12. Additional aggregate functions	225
6.7.5.13. COLLECT_LIST	225
6.7.5.14. COLLECT_SET	226
6.7.5.15. VARIANCE/VAR_POP	226
6.7.5.16. VAR_SAMP	227
6.7.5.17. COVAR_POP	227
6.7.5.18. COVAR_SAMP	228
6.7.6. Other functions	229
6.7.6.1. ARRAY	229
6.7.6.2. ARRAY_CONTAINS	229
6.7.6.3. CAST	230
6.7.6.4. COALESCE	230
6.7.6.5. DECODE	230
6.7.6.6. EXPLODE	231
6.7.6.7. GET_IDCARD_AGE	232

6.7.6.8. GET_IDCARD_BIRTHDAY	232
6.7.6.9. GET_IDCARD_SEX	232
6.7.6.10. GREATEST	233
6.7.6.11. INDEX	233
6.7.6.12. MAX_PT	234
6.7.6.13. ORDINAL	235
6.7.6.14. LEAST	235
6.7.6.15. SIZE	235
6.7.6.16. SPLIT	236
6.7.6.17. STR_TO_MAP	236
6.7.6.18. UNIQUE_ID	237
6.7.6.19. UUID	237
6.7.6.20. SAMPLE	237
6.7.6.21. CASE WHEN expression	238
6.7.6.22. IF	239
6.7.6.23. Additional functions	239
6.7.6.24. MAP	239
6.7.6.25. MAP_KEYS	240
6.7.6.26. MAP_VALUES	240
6.7.6.27. SORT_ARRAY	240
6.7.6.28. POSEXplode	241
6.7.6.29. STRUCT	241
6.7.6.30. NAMED_STRUCT	242
6.7.6.31. INLINE	242
6.7.6.32. BETWEEN AND expression	243
6.7.6.33. NVL	244
6.7.6.34. TABLE_EXISTS	244
6.7.6.35. PARTITION_EXISTS	245

6.8. UDFs	245
6.8.1. Overview	246
6.8.2. Types of parameters and returned values	246
6.8.3. UDFs	248
6.8.4. UDAFs	249
6.8.5. UDTFs	252
6.8.5.1. Overview	252
6.8.5.2. UDTF description	253
6.8.6. Python UDFs	256
6.8.6.1. Restricted environment	256
6.8.6.2. Third-party libraries	257
6.8.6.3. Types of parameters and returned values	257
6.8.6.4. UDFs	259
6.8.6.5. UDAFs	259
6.8.6.6. UDTFs	260
6.8.6.7. Reference resources	261
6.9. UDTs	262
6.9.1. Overview	263
6.9.2. Feature summary	263
6.9.3. Feature description	264
6.9.4. More examples	268
6.9.4.1. Example of using Java arrays	268
6.9.4.2. Example of using JSON	269
6.9.4.3. Example of using composite types	269
6.9.4.4. Example of aggregation	270
6.9.4.5. Example of using table-valued functions	270
6.9.5. Feature advantages	271
6.9.6. Performance advantages	271

6.9.7. Security advantages	271
6.10. UDJ	271
6.10.1. Overview	271
6.10.2. UDJ usage	272
6.10.2.1. Examples	272
6.10.2.2. Use Java to write the UDJ code	273
6.10.2.3. Create a UDJ function in MaxCompute	275
6.10.2.4. Use UDJ in MaxCompute SQL	275
6.10.2.5. Pre-sorting	277
6.10.3. Performance advantages	279
6.11. Parameterized view	280
6.12. Geographic functions	282
6.12.1. Usage notes	282
6.12.2. Constructors	283
6.12.2.1. ST_AsBinary	283
6.12.2.2. ST_AsGeoJson	283
6.12.2.3. ST_AsJson	283
6.12.2.4. ST_AsShape	283
6.12.2.5. ST_AsText	284
6.12.2.6. ST_GeomCollection	284
6.12.2.7. ST_GeomFromGeoJson	284
6.12.2.8. ST_GeomFromJSON	285
6.12.2.9. ST_GeomFromShape	285
6.12.2.10. ST_GeomFromText	285
6.12.2.11. ST_GeomFromWKB	285
6.12.2.12. ST_GeometryType	286
6.12.2.13. ST_LineString	286
6.12.2.14. ST_LineFromWKB	287

6.12.2.15. ST_MultiLineString	287
6.12.2.16. ST_MLineFromWKB	287
6.12.2.17. ST_MultiPoint	287
6.12.2.18. ST_MPointFromWKB	288
6.12.2.19. ST_MultiPolygon	288
6.12.2.20. ST_MPolyFromWKB	288
6.12.2.21. ST_Point	289
6.12.2.22. ST_PointFromWKB	289
6.12.2.23. ST_PointZ	289
6.12.2.24. ST_Polygon	289
6.12.2.25. ST_PolyFromWKB	290
6.12.2.26. ST_SetSRID	290
6.12.3. Accessors	290
6.12.3.1. ST_Area	290
6.12.3.2. ST_Centroid	291
6.12.3.3. ST_CoordDim	291
6.12.3.4. ST_Dimension	291
6.12.3.5. ST_Distance	292
6.12.3.6. ST_GeodesicLengthWGS84	292
6.12.3.7. ST_GeometryN	292
6.12.3.8. ST_Is3D	293
6.12.3.9. ST_IsClosed	293
6.12.3.10. ST_IsEmpty	293
6.12.3.11. ST_IsMeasured	294
6.12.3.12. ST_IsSimple	294
6.12.3.13. ST_IsRing	294
6.12.3.14. ST_Length	295
6.12.3.15. ST_M	295

6.12.3.16. ST_MaxM	295
6.12.3.17. ST_MinM	295
6.12.3.18. ST_X	296
6.12.3.19. ST_Y	296
6.12.3.20. ST_Z	296
6.12.3.21. ST_MaxX	297
6.12.3.22. ST_MaxY	297
6.12.3.23. ST_MaxZ	297
6.12.3.24. ST_MinX	297
6.12.3.25. ST_MinY	298
6.12.3.26. ST_MinZ	298
6.12.3.27. ST_NumGeometries	298
6.12.3.28. ST_NumInteriorRing	299
6.12.3.29. ST_NumPoints	299
6.12.3.30. ST_PointN	299
6.12.3.31. ST_StartPoint	300
6.12.3.32. ST_EndPoint	300
6.12.3.33. ST_SRID	300
6.12.4. Operations	300
6.12.4.1. ST_Aggr_ConvexHull	300
6.12.4.2. ST_Aggr_Intersection	301
6.12.4.3. ST_Aggr_Union	301
6.12.4.4. ST_Bin	301
6.12.4.5. ST_BinEnvelope	301
6.12.4.6. ST_Boundary	302
6.12.4.7. ST_Buffer	302
6.12.4.8. ST_ConvexHull	302
6.12.4.9. ST_Difference	302

6.12.4.10. ST_Envelope	303
6.12.4.11. ST_ExteriorRing	303
6.12.4.12. ST_InteriorRingN	303
6.12.4.13. ST_Intersection	304
6.12.4.14. ST_SymmetricDiff	304
6.12.4.15. ST_Union	305
6.12.5. Relationship tests	305
6.12.5.1. ST_Contains	305
6.12.5.2. ST_Crosses	305
6.12.5.3. ST_Disjoint	306
6.12.5.4. ST_EnvIntersects	306
6.12.5.5. ST_Equals	306
6.12.5.6. ST_Intersects	307
6.12.5.7. ST_Overlaps	307
6.12.5.8. ST_Relate	308
6.12.5.9. ST_Touches	308
6.12.5.10. ST_Within	308
6.12.6. Geohash index functions	309
6.12.6.1. ST_GeoHash	309
6.12.6.2. ST_PointFromGeoHash	309
6.12.6.3. ST_EnvelopeFromGeoHash	309
6.12.6.4. ST_GeoHashNeighbours	310
6.12.7. S2 mesh functions	310
6.12.7.1. ST_S2CellIdsFromGeom	310
6.12.7.2. ST_S2CellIdsFromText	310
6.12.7.3. ST_S2CellCenterPoint	311
6.12.7.4. ST_S2CellNeighbours	311
6.12.8. Geodesic functions	311

6.12.8.1. ST_AreaWGS84	311
6.12.8.2. ST_DistanceWGS84	311
6.12.8.3. ST_BufferWGS84	312
6.12.8.4. ST_GeodesicDistance	312
6.12.8.5. ST_Distance_Sphere	312
6.12.8.6. ST_Area_Sphere	313
6.12.9. R-tree index functions	313
6.12.9.1. ST_BuildRTreeIndex	313
6.12.9.2. ST_ContainsFromRTree	314
6.12.9.3. ST_CrossesFromRTree	314
6.12.9.4. ST_EqualsFromRTree	314
6.12.9.5. ST_IntersectsFromRTree	314
6.12.9.6. ST_OverlapsFromRTree	314
6.12.9.7. ST_TouchesFromRTree	315
6.12.9.8. ST_WithinFromRTree	315
6.12.9.9. ST_KNNFromRTree	315
6.12.9.10. Example	315
6.12.10. Other functions	317
6.12.10.1. ST_IsValid	317
6.12.10.2. ST_Transform	317
6.13. SQL Function	318
6.14. CLONE TABLE	318
6.15. MaxCompute Hash Clustering	321
6.15.1. Background information	321
6.15.2. Descriptions	323
6.15.2.1. Enable or disable Hash Clustering	323
6.15.2.2. Create a hash clustering table	323
6.15.2.3. Modify table attributes	325

6.15.2.4. View and verify table attributes	325
6.15.3. Benefits	326
6.15.3.1. Bucket pruning and index optimization	326
6.15.3.2. Aggregation optimization	326
6.15.3.3. Storage optimization	327
6.15.4. ShuffleRemove	328
6.15.5. Limits	328
6.16. MaxCompute SQL limits	329
6.17. Common MaxCompute SQL parameter settings	330
6.17.1. MAP configurations	330
6.17.2. JOIN configurations	331
6.17.3. Reduce configurations	331
6.17.4. UDF configurations	332
6.17.5. MAPJOIN configurations	332
6.17.6. Configure data skew	333
6.18. MapReduce-to-SQL conversion for execution	333
6.18.1. Overview	333
6.18.2. Configure local running settings	334
6.18.3. Operation settings in DataWorks	334
6.18.4. View running details	335
6.18.5. Perform operations on the distributed file system	337
6.19. Analysis of the mapping between SQL input and output ...	337
6.19.1. Features	338
6.19.2. Usage notes	338
6.20. Common MaxCompute SQL errors and solutions	339
6.20.1. Data skew	339
6.20.1.1. Overview	339
6.20.1.2. GROUP BY skew	340

6.20.1.3. DISTRIBUTE BY skew	340
6.20.1.4. JOIN skew	340
6.20.1.5. MULTI-DISTINCT skew	341
6.20.1.6. Data skew caused by misuse of dynamic partitioni...	341
6.20.2. Quota and resource usage	341
6.20.3. MaxCompute storage optimization tips	343
6.20.4. UDF OOM error	344
6.21. Appendix	345
6.21.1. Escape character	345
6.21.2. LIKE matching	346
6.21.3. Regular expressions	346
6.21.4. Reserved words	347
6.21.5. New data type settings	348
7. MaxCompute Tunnel	349
7.1. Overview	349
7.2. Tunnel service connections	349
7.3. Selection of cloud data migration tools	350
7.4. Introduction to the tools	350
7.5. Tunnel SDK overview	351
7.5.1. Overview	352
7.5.2. TableTunnel	352
7.5.3. InstanceTunnel	354
7.5.4. UploadSession	354
7.5.5. DownloadSession	356
7.5.6. TunnelBufferedWriter	357
7.6. Tunnel SDK example	358
7.6.1. Simple upload example	358
7.6.2. Simple download example	360

7.6.3. Multithread upload example	361
7.6.4. Multithread download example	364
7.6.5. Example of uploading data by using BufferedWriter	366
7.6.6. Example of uploading data by using BufferedWriter in ...	366
7.6.7. Examples of uploading and downloading complex data	367
7.7. Appendix	370
7.7.1. Tunnel upload/download FAQ	370
7.7.2. Common tunnel error codes	372
8. MaxCompute MapReduce	375
8.1. Overview	375
8.1.1. MapReduce	375
8.1.2. Extended MapReduce	376
8.1.3. Open-source compatibility with MapReduce	377
8.2. Features	382
8.2.1. Run command	382
8.2.2. Concepts	384
8.2.2.1. MapReduce	384
8.2.2.2. Sorting	384
8.2.2.3. Partition	384
8.2.2.4. Combiner	384
8.2.2.5. Submit a job	384
8.2.2.6. Input and output	386
8.2.2.7. Read data from resources	386
8.2.2.8. Run MapReduce tasks locally	386
8.3. SDK introduction	390
8.3.1. Major API overview	390
8.3.2. API description	390
8.3.2.1. MapperBase	390

8.3.2.2. ReducerBase	391
8.3.2.3. TaskContext	391
8.3.2.4. JobConf	392
8.3.2.5. JobClient	393
8.3.2.6. RunningJob	394
8.3.2.7. InputUtils	394
8.3.2.8. OutputUtils	394
8.3.2.9. Pipeline	395
8.3.3. Compatibility with Hadoop MapReduce	396
8.4. Data types	411
8.5. Limits	412
8.6. Sample programs	412
8.6.1. WordCount example	412
8.6.2. MapOnly example	414
8.6.3. Example: Input and output data to multiple objects	415
8.6.4. Multi-task example	418
8.6.5. Secondary sorting example	420
8.6.6. Resource usage example	422
8.6.7. Example for using counters	423
8.6.8. grep example	425
8.6.9. JOIN example	428
8.6.10. Sleep example	430
8.6.11. unique example	434
8.6.12. Sort example	436
8.6.13. Example of using partitioned table as an input	437
8.6.14. Pipeline example	439
9. MaxCompute Graph	441
9.1. Graph overview	441

9.1.1. Graph overview	441
9.1.2. Graph data structure	441
9.1.3. Graph logic	442
9.1.3.1. Load graph	442
9.1.3.2. Iterative computation	443
9.1.3.3. End of iteration	443
9.1.4. Aggregator overview	444
9.2. Graph feature overview	451
9.2.1. Run a job	452
9.2.2. Input and output	453
9.2.3. Read data from resources	454
9.2.3.1. Add resource in Graph program	454
9.2.3.2. Use resources in Graph	454
9.3. Graph SDK introduction	455
9.4. Development and debugging	456
9.4.1. Development procedure	456
9.4.2. Development example	456
9.4.3. Local debugging	457
9.4.4. Temporary directory for local jobs	459
9.4.5. Cluster debugging	460
9.4.6. Performance optimization	460
9.4.6.1. Configure job parameters	460
9.4.6.2. Use Combiner	461
9.4.6.3. Reduce data input	461
9.4.6.4. JAR packages	462
9.5. Application limits	462
9.6. Sample programs	463
9.6.1. SSSP	463

9.6.2. PageRank	465
9.6.3. K-means clustering	467
9.6.4. BiPartiteMatching	472
9.6.5. Strongly-connected component	475
9.6.6. Connected component	482
9.6.7. Topological sorting	485
9.6.8. Linear regression	487
9.6.9. Count triangles	492
9.6.10. GraphLoader	494
10. Java SDK	501
11. PyODPS	502
11.1. Overview	502
11.2. Quick start	502
11.3. Installation instructions	503
11.4. Platform instructions	504
11.4.1. Overview	504
11.4.2. Use local PyODPS	504
11.4.3. Use PyODPS in DataWorks	504
11.5. Basic operations	506
11.5.1. Overview	506
11.5.2. Projects	506
11.5.3. Tables	507
11.5.4. SQL	514
11.5.5. Task instances	517
11.5.6. Resources	518
11.5.7. Functions	520
11.6. DataFrame	521
11.7. User experience enhancement	526

11.7.1. Command line	527
11.7.2. IPython	528
11.7.3. Jupyter Notebook	531
11.8. Configuration	534
11.9. API overview	537
11.10. FAQ	537
12. Java sandbox limits	540
13. Volume lifecycle management	544
13.1. Overview	544
13.2. Volume lifecycle operations	544
14. Spark on MaxCompute	545
14.1. Overview	545
14.2. Project resources	545
14.3. Environment settings	545
14.3.1. Decompress the Spark on MaxCompute release packag...	545
14.3.2. Set environment variables	546
14.3.3. Configure Spark-defaults.conf	546
14.4. Quick start	547
14.5. Demo	549
14.6. Common cases	551
14.6.1. WordCount example	551
14.6.2. OSS access example	551
14.6.3. MaxCompute table read/write example	552
14.6.4. MaxCompute Table Spark-SQL example	554
14.6.5. MaxCompute self-developed Console mode example	555
14.6.6. MaxCompute Table PySpark example	556
14.6.7. Mllib example	557
14.6.8. PySpark interactive execution example	558

14.6.9. Spark-shell interactive execution example (read tables)	558
14.6.10. Spark-shell interactive execution example (MLlib and...	558
14.6.11. SparkR interactive execution example	559
14.6.12. GraphX-PageRank example	560
14.6.13. Spark Streaming - NetworkWordCount example	562
14.7. Maven dependencies	562
14.8. Special notes	564
14.8.1. Running modes	564
14.8.2. Streaming tasks	565
14.8.3. Job diagnosis	565
14.9. APIs supported by Spark	567
14.9.1. Spark Shell	567
14.9.2. Spark R	568
14.9.3. Spark SQL	568
14.9.4. Spark JDBC	568
14.10. Spark dynamic resource allocation	569
14.11. FAQ	570
15. Elasticsearch on Maxcompute	572
15.1. Overview	572
15.2. Workflow	572
15.2.1. Overview	572
15.2.2. Distributed retrieval workflow	572
15.2.3. Full-text retrieval process	573
15.2.4. Authentication process	574
15.3. Quick start	574
15.4. Support for Elasticsearch applications	575
15.4.1. ElasticSearch typical practice	575
15.4.2. Elasticsearch on MaxCompute support for VPC	576

15.5. Special notes	576
15.5.1. Find the Elasticsearch service domain name	576
15.5.2. Import table data from MaxCompute to Elasticsearch	577
16.Flink on MaxCompute	579
17.Non-structured data access and processing (integrated compu... ..	581
17.1. Overview	581
17.2. Internal data sources	581
17.2.1. OSS data source	581
17.2.1.1. Preface	581
17.2.1.2. Use the built-in extractor to read OSS data	581
17.2.1.2.1. Overview	581
17.2.1.2.2. Create an external table	582
17.2.1.2.3. Query an external table	582
17.2.1.2.4. MSCK REPAIR TABLE	583
17.2.1.3. Custom extractors	584
17.2.1.3.1. Overview	584
17.2.1.3.2. Define StorageHandler	584
17.2.1.3.3. Define an extractor	584
17.2.1.3.4. Compile and package code	585
17.2.1.3.5. Create an external table	586
17.2.1.3.6. Query an external table	586
17.2.1.4. Advanced usage	587
17.2.1.4.1. Use a custom extractor to read external unstruc... ..	587
17.2.1.5. Data partitions	589
17.2.1.5.1. Overview	589
17.2.1.5.2. Standard organization method and path forma... ..	590
17.2.1.5.3. Custom path of partition data in OSS	591
17.2.1.5.4. Access fully-customized non-partitioned data su... ..	592

17.2.1.6. Output OSS data	592
17.2.1.6.1. Create an external table	592
17.2.1.6.2. Write data to a TSV text file by using an INSE...	593
17.2.1.6.3. Write data to an unstructured file by using an...	594
17.2.1.6.4. Migrate data between different storage media ...	594
17.2.1.7. STS mode authorization for OSS	595
17.2.2. Table Store data source	597
17.2.2.1. Preface	597
17.2.2.2. MaxCompute reads and computes data in Table St...	598
17.2.2.2.1. Prerequisites and assumptions	598
17.2.2.2.2. Create an external table	598
17.2.2.2.3. Access Table Store data through an external ta...	599
17.2.2.3. Export data from MaxCompute to Tablestore	600
17.2.3. AnalyticDB data source	601
17.2.3.1. Overview	601
17.2.3.2. Write data to AnalyticDB	601
17.2.3.2.1. Create an external table	601
17.2.3.2.2. Write and query data	602
17.2.3.3. Read data from AnalyticDB	602
17.2.4. RDS data source	602
17.2.4.1. Overview	602
17.2.4.2. Write data to RDS	603
17.2.4.2.1. Create an external table	603
17.2.4.2.2. Write and query data	603
17.2.4.3. Read data from RDS	603
17.2.5. HDFS data source (Alibaba Cloud)	604
17.2.5.1. Overview	604
17.2.5.2. Data processing for common tables	604

17.2.5.2.1. Write data to HDFS	604
17.2.5.2.1.1. Create an external table	604
17.2.5.2.1.2. Write and query data	605
17.2.5.2.2. Read data from HDFS	605
17.2.5.3. Data processing for partitioned tables	606
17.2.6. TDDL data source	607
17.2.6.1. Overview	607
17.2.6.2. Prerequisites	608
17.2.6.3. Create a TDDL external table	608
17.2.6.3.1. Syntax	608
17.2.6.3.2. Example	611
17.2.6.4. Read data from an external table	612
17.2.6.5. Write data to an external table in the append mo...	613
17.3. External data sources	614
17.3.1. HDFS data source (open-source)	614
17.3.1.1. Overview	614
17.3.1.2. Write data to HDFS	614
17.3.1.2.1. Create an external table	614
17.3.1.2.2. Write and query data	615
17.3.1.3. Read data from HDFS	615
17.3.2. MongoDB data source	615
17.3.2.1. Overview	615
17.3.2.2. Prerequisites	615
17.3.2.3. Write data to MongoDB	616
17.3.2.3.1. Create an external table	616
17.3.2.3.2. Write and query data	617
17.3.2.4. Read data from MongoDB	617
17.3.3. HBase data source	617

17.3.3.1. Overview	617
17.3.3.2. Write data to HBase	617
17.3.3.2.1. Create an external table	618
17.3.3.2.2. Write and query data	618
17.3.3.3. Read data from HBase	618
18.Unstructured data access and processing (inside MaxCompute)	620
18.1. Overview	620
18.2. Create a volume external table	620
18.2.1. Syntax	620
18.2.2. Use the built-in StorageHandler to create an external...	621
18.2.3. Use a custom StorageHandler to create a table	622
18.3. Access a volume external table	623
19.Multi-region cluster deployment on MaxCompute	624
19.1. Overview	624
19.2. Characteristics of multi-region deployment	624
19.3. Instructions on multi-region deployment	625
19.4. Multi-region deployment examples	625
19.4.1. Synchronize table data among multiple clusters	626
19.4.2. Query the status of data synchronization between pri...	627
19.4.3. Cross-region direct read	628
19.4.4. Cross-region JOIN	630
20.Security solution	632
20.1. Target users	632
20.2. Quick start	632
20.3. User authentication	635
20.4. Project user and authorization management	635
20.4.1. Overview	635
20.4.2. User management	635

20.4.3. Role management	636
20.4.4. ACL authorization actions	636
20.4.5. View permissions	639
20.5. Cross-project resource sharing	640
20.5.1. Overview	640
20.5.2. Package usage	641
20.5.2.1. Operations for package creators	641
20.5.2.2. Operations for package users	642
20.6. Project protection	643
20.6.1. Overview	643
20.6.2. Data protection	643
20.6.3. Data export methods when project protection is enab... ..	644
20.6.4. Resource sharing and data protection	645
20.7. Project security configuration	646
20.8. Authorization policies	646
20.8.1. Policy overview	646
20.8.2. Policy-related terms	648
20.8.3. Access policy structure	649
20.8.3.1. Overview	649
20.8.3.2. Authorization statement structure	650
20.8.3.3. Conditional block structure	650
20.8.3.4. Conditional action type	651
20.8.3.5. Conditional keywords	651
20.8.4. Access policy norm	652
20.8.4.1. Principal naming convention	652
20.8.4.2. Resource naming convention	652
20.8.4.3. Action naming	653
20.8.4.4. Condition keys naming	654

20.8.4.5. Access policy example	654
20.8.5. Differences between policy authorization and ACL aut... ..	655
20.8.6. Application limits	655
20.9. Collection of security statements	656
20.9.1. Project security configuration	656
20.9.2. Project permission management	657
20.9.3. Package-based resource sharing	658
21. Frequently-used tools	660
21.1. MaxCompute console	660
21.1.1. Usage notes	660
21.1.2. Install the client	660
21.1.3. Configuration description	661
21.2. Eclipse development plugin	664
21.2.1. Install Eclipse	664
21.2.2. Create a project	668
21.2.2.1. Method 1	668
21.2.2.2. Method 2	670
21.2.3. MapReduce running example	672
21.2.3.1. Quickly run a WordCount example	672
21.2.3.2. Run a custom MapReduce program	675
21.2.4. UDF development and running example	689
21.2.4.1. Local debug UDF programs	689
21.2.4.1.1. Run a UDF from the menu bar	689
21.2.4.1.2. Use the right-click shortcut menu to quickly ru... ..	690
21.2.4.2. Run a UDF program	692
21.2.5. Graph running example	694
22. MaxCompute FAQ	698
23. Open source features of MaxCompute	705

1. What is MaxCompute?

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

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

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

MaxCompute is designed to provide an intuitive approach to analyze and process large amounts of data without the need for distributed computing knowledge. MaxCompute is widely implemented within Alibaba's businesses for scenarios such as 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 channel
 - Tunnel: provides highly-concurrent offline data upload and download services. MaxCompute Tunnel enables you to upload or download a large volume of data to or from MaxCompute. You must use a Java programming API to access MaxCompute Tunnel.
 - DataHub: provides real-time upload and download services. Data uploaded through DataHub is available immediately, while data uploaded through 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 far more powerful and is capable of processing 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 migrated seamlessly 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 having a deep understanding 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 using MapReduce. MapReduce provides a Java programming interface.
 - 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. You can edit and evolve a graph through iteration to obtain the final result. Typical applications include [PageRank](#), [single source shortest path \(SSSP\) algorithm](#), and [K-means clustering algorithm](#).

- Unstructured data access and processing (integrated computing scenarios): MaxCompute SQL cannot directly process external data (such as unstructured data from 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 resolve this problem.

MaxCompute can process the following data sources by creating external tables:

- Internal data sources: OSS, Table Store, AnalyticDB, ApsaraDB for RDS, HDFS (Alibaba Cloud), and TDDL.
- External data sources: HDFS (open source), MongoDB, and Hbase.
- Unstructured data access and processing (inside MaxCompute): By reading and writing volumes, MaxCompute can store unstructured data, which otherwise must be stored in an external storage system.
- Spark on MaxCompute: a big data analytics engine designed by Alibaba Cloud to provide big data processing capabilities for Alibaba, 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 volumes of data. It provides near-real-time (NRT) search performance for government agencies and enterprises. For more information, see [Elasticsearch on MaxCompute](#).
- SDK: a toolkit provided for developers. For more information, see [Java SDK](#).
- Security solution: MaxCompute provides powerful security services to guarantee user data security.

2.Usage notes

You can selectively read topics in this document based on your requirements. This topic provides reading suggestions in the document based on user skill level.

Beginners

If you are a beginner, we recommend that you read the following topics:

- **What is MaxCompute:** The topic provides a general introduction of MaxCompute and its core features. You can obtain general knowledge about MaxCompute.
- **Quick start:** The topic provides step-by-step examples and instructions on how to perform basic MaxCompute operations, such as installing and configuring the client, creating tables, granting permissions, importing and exporting data, running SQL tasks, running user-defined functions (UDFs), and running MapReduce.
- **Basic concepts and common commands:** The topic introduces the basic concepts of MaxCompute and common MaxCompute commands. This topic helps familiarize yourself with MaxCompute operations.
- **Tools:** The topic describes how to download, configure, and use common MaxCompute tools to perform data analysis.

Data analysts

If you are a data analyst, we recommend that you read the following topics:

MaxCompute SQL: The topic describes how to query and analyze large amounts of data stored in MaxCompute. This topic covers the following operations:

- Execute DDL statements CREATE, DROP, and ALTER to manage tables and partitions.
- Execute SELECT statements to select records in a table, and execute WHERE clauses to view the records that meet a specified filtering condition.
- Associate two tables through an EQUIJOIN operation.
- Execute GROUP BY statements to aggregate columns.
- Execute INSERT OVERVIEW or INSERT INTO statements to insert results into another table.
- Use built-in functions and UDFs to complete a variety of computations.
- Use UDTs to reference classes or objects of third-party languages in SQL statements to obtain data or call methods.
- Use UJEs to implement flexible cross-table and multi-table custom operations, and reduce the operations on the underlying details of the distributed system through MapReduce.
- Use the Select Transform feature to simplify the reference of script code.
- Collect table statistics and configure table lifecycles.
- Use regular expressions.

Developers

If you are an experienced developer with basic understanding of distributed computing and need to perform data analysis that cannot be implemented with SQL, we recommend that you read the following topics on advanced MaxCompute functional modules:

- **MapReduce** is a Java programming model provided by MaxCompute. You can use the API to write MapReduce programs and process MaxCompute data.

- MaxCompute Graph is a processing framework designed that iteratively computes and models graphs. A graph consists of vertices and edges, both of which contain values. MaxCompute Graph iteratively edits and evolves graphs to obtain analysis results.
- Eclipse plugin provides an IDE to help you complete development of MapReduce, UDFs, and Graph.
- Java SDK is a toolkit provided to developers.
- MaxCompute Tunnel allows you to perform batch upload and download operations on offline data to and from MaxCompute.

Project owners or administrators

If you are a project owner or administrator, we recommend that you read the following topics:

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

3. Preparations

3.1. Log on to the ASCM console

This topic describes how to log on to the Apsara Stack Cloud Management (ASCM) console.

Prerequisites

- The IP address or domain name of the ASCM console is obtained from deployment personnel. The URL used to log on to the ASCM console is in the following format: `https://IP address or domain name of the ASCM console`.
- A browser is available. We recommend that you use Google Chrome.

Procedure

1. In the address bar, enter the URL used to log on to the ASCM console. Press Enter.
2. Enter the correct username and password.

Obtain the username and password used to log on to the console from the operations administrator.

Note When you log on to the ASCM console for the first time, you must change the password as prompted. For security purposes, your password must meet the complexity requirements. Specifically, the password must be 8 to 20 characters in length and must contain at least two of the following character types: uppercase letters, lowercase letters, digits, and special characters such as ! @ # \$ %

3. Click **Login** to go to the homepage of the ASCM console.

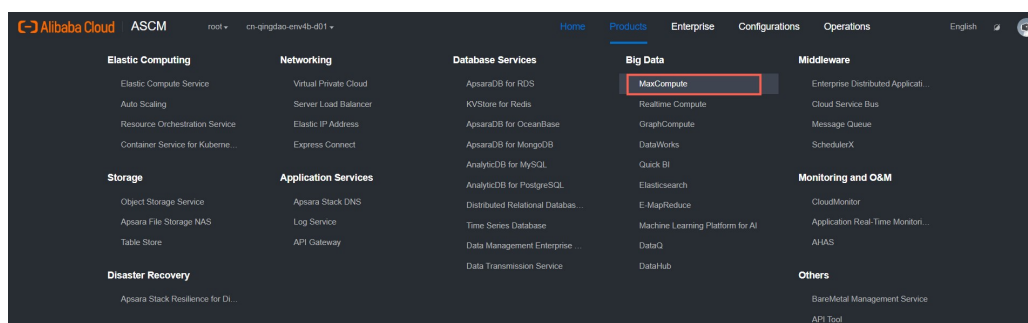
3.2. Create an Apsara Stack tenant account

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

Procedure

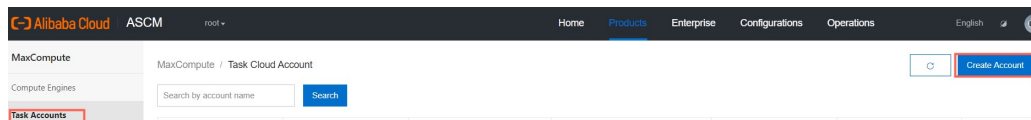
1. Log on to the Apsara Stack Cloud Management (ASCM) console as an administrator.
2. In the top navigation bar, choose **Products > Big Data > MaxCompute** to go to the **MaxCompute** homepage.

Go to the MaxCompute homepage



3. In the left-side navigation pane, click **Task Accounts**. In the upper-right corner of the page that appears, click **Create Account**.

Create an account



4. In the dialog box that appears, specify the required parameters and click OK.

Account configurations

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.

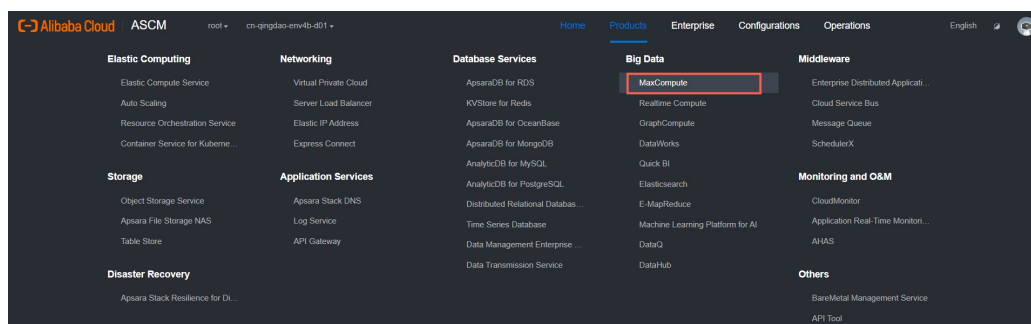
3.3. Create a project

This topic describes how to create a MaxCompute project.

Procedure

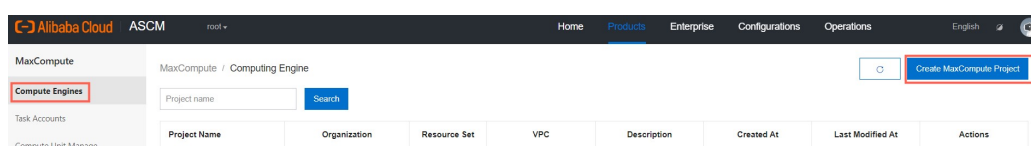
1. Log on to the Apsara Stack Cloud Management (ASCM) console as an administrator.
2. In the top navigation bar, choose **Products > Big Data > MaxCompute** to go to the **MaxCompute** homepage.

Go to the MaxCompute homepage



3. In the left-side navigation pane, click **Computing Engines**. In the upper-right corner of the page that appears, click **Create MaxCompute Project**.

Create a MaxCompute project



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

Region configurations

Parameter	Description
Organization	The organization that you select for the project.
Resource Set	The resource set in the selected organization.
Region	The region where the cluster is deployed.
VPC	The Virtual Private Cloud (VPC) in the region. The VPC is used for network isolation.
Cluster	The information about the cluster of the project.
Encryption Settings	No is selected for Encrypt by default. If you select Yes for Encrypt, the available encryption algorithms are AES-CTR, AES-256, and RC4.
Encryption Key	If you select Yes for Encrypt, this parameter is available. The default value is Generate Automatically.

Resource configurations

Parameter	Description
Resource Group	The resource group in the selected cluster. If no resource groups are available, create one.
Apply (GB)	The storage space that is requested for the project. The value cannot be greater than the total capacity of the disk.

Basic configurations

Parameter	Description
MaxCompute Project Name	The name of the project.
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.
Access ID	The AccessKey ID of the Apsara Stack tenant account that corresponds to the project owner. You cannot specify this parameter.
Access Key	The AccessKey secret of the Apsara Stack tenant account that corresponds to the project owner. You cannot specify this parameter.

Parameter	Description
Task Account	Optional. The account of the task.
Access ID	The AccessKey ID of the task account. You cannot specify this parameter.
Access Key	The AccessKey secret of the task account. You cannot specify this parameter.

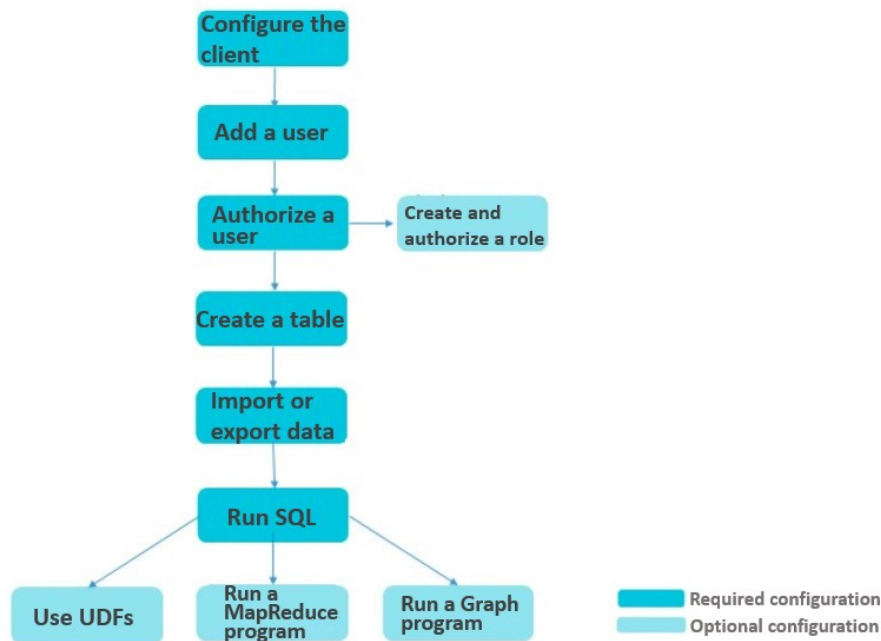
4. Quick start

4.1. Overview

This topic describes the operation process of MaxCompute. It aims to provide you with step-by-step instructions on basic MaxCompute operations.

MaxCompute operation process shows the procedure.

MaxCompute operation process



1. **Configure the client.**

You must install and configure the **client** to use all the features of MaxCompute.

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 only perform operations on the project after the user is authorized.

4. (Optional) **Create and authorize a role.**

It can be very time-consuming to individually authorize users if a project 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 assigned this role are granted the same permissions.

5. **Create a table.**

After you are added to a project and authorized, you can start to 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 compile your own Java tools and import and export data.

7. Run SQL.

Only the limits on a few common SQL statements are described here. For more information about how to execute SQL statements, see [MaxCompute SQL](#).

8. Then, you can perform any of the following optional operations:

- [Use UDFs](#).

After you install the MaxCompute client, you can try to use UDFs. MaxCompute provides three types of UDFs: UDFs, UDAFs, and UDTFs. These functions are collectively known as UDFs.

- [Run a MapReduce program](#).

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

- [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 install, configure, and run the client.

Context

The client is developed in Java. Make sure that you have JRE 1.8 installed on your local PC. In addition, make sure that you have an Apsara Stack tenant account and have obtained the AccessKey ID and AccessKey secret.

 **Note** Before you configure the 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 computer.
2. Decompress the package. The package contains the following four folders:

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

3. Edit the following information in the odps_config.ini file under the conf folder:

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

 Note

- Set `access_id` to the AccessKey ID and `access_key` to the AccessKey secret of your Apsara Stack tenant account.
- `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 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.
- For more information about the client, see [MaxCompute client](#).

4. After the modification, run the `./bin/odpscmd` program in a Linux system or `./bin/odpscmd.bat` in a Windows system in the bin directory to execute SQL statements. 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 users

Other than the project owner, all other users must be added to a MaxCompute project and granted the corresponding permissions before they can perform any operations on the project. This topic describes how a project owner can add or delete users in a project.

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

Add users

Run the following command to add a user:

```
ADD USER <full_username>;
```

Run the following command on the client to add a user (bob@aliyun.com) to a project:

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

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

```
LIST USERS;
```

 Note

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

Delete users

Run the following command to delete a user:

```
REMOVE USER <full_username>;
```

Run the following command on the client to delete a user from a project:

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

 Note

- Before you delete a user, make sure that you have revoked all the permissions of the user. If you delete a user without revoking the permissions of the user, the permissions are retained. If the user is added to the project again, the user will have the permissions that were granted previously.
- For more information about how to add or delete users, see [Manage users in a project](#).

4.4. Grant and view permissions

4.4.1. Overview

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

Authorization is a process of granting the permission to perform an operation (such as reading, writing, or viewing), on objects (such as tables, tasks, and resources) in MaxCompute.

This topic is intended for project administrators. If you are a regular MaxCompute user, verify that you have obtained the required permissions. You can quickly skim this topic.

MaxCompute provides two authorization mechanisms, [ACL authorization](#) and [policy authorization](#).

4.4.2. ACL authorization

This topic describes the commands for ACL authorization and provides examples.

ACL authorization in MaxCompute applies to the following objects: project, table, function, resource, instance, and task. Every object has different operation permissions. For more information, see [ACL authorization actions](#).

Command syntax:

```
GRANT privileges ON project_object TO project_subject
REVOKE privileges ON project_object FROM project_subject
privileges ::= action_item1, action_item2, ...
project_object ::= PROJECT project_name | TABLE schema_name |
INSTANCE inst_name | FUNCTION func_name |
RESOURCE res_name | JOB job_name
project_subject ::= USER full_username | ROLE role_name
```

 **Note** You can skip the ROLE clause in the preceding command. It is described in the topics after this.

Example:

```
grant CreateTable on PROJECT $user_project_name to USER ALIYUN$bob@aliyun.com;
-- Grant bob@aliyun.com the permissions to create tables in the project named $user_project_name.
grant Describe on Table $user_table_name to USER ALIYUN$bob@aliyun.com;
-- Grant bob@aliyun.com the permissions to obtain information (Describe permission) in the table named $$user_table_name.
grant Execute on Function $user_function_name to USER ALIYUN$bob@aliyun.com;
-- Grant bob@aliyun.com the permissions to run the function named $user_function_name.
```

4.4.3. Policy authorization

This topic describes the commands for policy authorization and provides an example.

Policy authorization is a principal-based process. For more information, see [Authorization policies](#).

Command syntax:

```
GET POLICY;
PUT POLICY <policyFile>;
GET POLICY ON ROLE <roleName>;
PUT POLICY <policyFile> ON ROLE <roleName>;
```

Example:


```
{
  "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 a user the permissions to perform list, read, describe, and select operations on the project and table a in the project.

4.4.4. View permissions

You can run a command to view user permissions in MaxCompute.

Run the following command to view the permissions of a user:

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

 **Note** For more information about how to view user permissions, 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 categorize users with the same permissions. After you authorize a role, all users assigned with this role are granted the same permissions. This topic describes how to create a role and grant permissions to it.

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

Create a role

Run the following command to create a role:

```
CREATE ROLE <roleName>;
```

Example:

```
create role player;
```

Add a user to a role

Run the following command to add a user to a role:

```
GRANT <roleName> TO <full_username>;
```

Example:

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

Delete a role

Run the following command to delete a role:

```
DROP ROLE <roleName>;
```

Example:

```
drop role player;
```

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

Grant permissions to a role

The command for granting permissions to a role is similar to the command for granting permissions to a user. For more information about how to grant permissions to a user, see [Grant and view permissions](#). 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.

Run the following command to create a table:

```
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]
CREATE TABLE [IF NOT EXISTS] table_name
LIKE existing_table_name
```

Example:

```
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 with a lifecycle.
create table test4 like test3;
-- Table test3 has the same attributes (such as the field type and partition type) as those
of test4, except for lifecycle.
create table test5 as select * from test2;
-- Create table test5 without replicating the partition and lifecycle information of test2
to it. Only data of test2 is copied to test5.
```

You can set partitions or 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 the lifecycle, see [Modify the lifecycle of a table](#).

4.6.2. Obtain table information

This topic describes how to obtain the table information.

Command syntax:

```
desc <table_name>;
```

Example:

```
desc test3;
-- Obtain the information about test3.
desc test4;
-- Obtain the information about test4.
```

4.6.3. Delete a table

This topic describes how to delete a table.

Run the following command to delete a table:

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

Example:

```
drop table test2;
```

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

4.7. Import or export data

You can compile your own Java tools by using the SDK provided by MaxCompute Tunnel to import or export data. For the sample code, see [Tunnel SDK examples](#).

4.8. Run SQL

4.8.1. Overview

This topic describes the limits to a few common SQL statements only. For more information about how to run SQL statements, see [MaxCompute SQL](#).

Note the following issues when using 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 seamlessly 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 statement

This topic describes limits of the SELECT statement.

The following limits apply to the SELECT statement:

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

```
select substr(col2, 2) from tbl group by substr(col2, 2);
-- Allowed: The key of the GROUP BY statement is an expression of columns in the input table.
select col2 from tbl group by substr(col2, 2);
-- Not allowed: The key of the GROUP BY statement is not included in columns of the SELECT statement.
select substr(col2, 2) as c from tbl group by c;
-- Not allowed: The key of the GROUP BY statement is the alias of a column, or output column of the SELECT statement.
```

 **Note** For SQL parsing, the GROUP BY operation is conducted before the SELECT operation, which means the GROUP BY statement can only use the column or expression of the input table as the key.

- DISTRIBUTE BY must be added in front of SORT BY.
- The key of ORDER BY/SORT BY/DISTRIBUTE BY must be the output column of SELECT statement, or the column alias.

```
select col2 as c from tbl order by col2 limit 100
-- Not allowed: The key of the ORDER BY statement is not the output column of the SELECT statement, or the column alias.
select col2 from tbl order by col2 limit 100;
-- Allowed: If an output column of the SELECT statement does not have an alias, the column name is used as the alias.
```

 **Note** For SQL parsing, the ORDER BY, SORT BY, and DISTRIBUTE BY operations come after the SELECT operation. Therefore, they can only accept the output columns of the SELECT statement as the key.

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

4.8.3. INSERT statement

This topic describes the limits of the INSERT statement.

The following limits apply to INSERT statements:

- When an INSERT statement is used to insert data into a partition, the partition column cannot be included in the select list.

```
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 columns and are not allowed in a INSERT statement for a static partition.
```

- When an INSERT statement is used to insert a dynamic partition, the dynamic partition column must be included in the select list.

```
insert overwrite table sale_detail_dypartition (sale_date='2017', region) select shop_name, customer_id, total_price from sale_detail;
-- An error is returned. When a dynamic partition is specified for the insert, the dynamic partition columns must be included among the selected columns.
```

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

4.8.4. JOIN statement

This topic describes the limits of JOIN statements.

The following limits apply to JOIN operations:

- 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 operations, see [JOIN statement](#).

4.8.5. Other limits

This topic describes the other application limits of MaxCompute SQL.

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

4.9. Compile and use UDFs

4.9.1. Overview

This topic provides examples on how to compile and use MaxCompute UDFs.

MaxCompute provides three types of UDFs: UDFs, UDAFs, and UDTFs. These functions are collectively known as UDFs.

Note

- UDFs only support Java APIs. To compile a UDF program, you can upload UDF code to your project by adding resources, and run the CREATE FUNCTION statement to create a UDF.
- This topic provides examples of UDF, UDAF, and UDTF code.

4.9.2. UDF example

This topic uses the convert-to-lowercase function as an example to demonstrate the process of creating a UDF. Specifically, follow these steps:

Procedure

1. Write code.

To archive function, write a program and compile in terms of MaxCompute UDF frame.

```
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 running UDF. User code must be added to MaxCompute by adding resources. Java UDFs must be compiled into the JAR package and added in MaxCompute as a JAR resource. The UDF framework loads the JAR package automatically and runs UDF.

Example of the command to add JAR resources:

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

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

3. Register the UDF.

When your JAR package is uploaded, MaxCompute does not have any information about this UDF. Therefore, you must register a unique function name in MaxCompute, and specify to which function and under which JAR resources this function name corresponding.

An example of using commands to register the UDF is as follows:

```
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 an example of UDAF code for your reference.

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

The following UDAF code is for reference only:


```

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 UDTF code for your reference.

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

UDTF code 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);
}
}
}

```

4.10. Compile and run a MapReduce job

This topic describes how to quickly run a MapReduce program after the MaxCompute client is installed.

Context

A WordCount program is used as an example in this topic.

Before you compile and run a MapReduce program, make sure that the following requirements are met:

- JDK 1.8 has been installed on your host.
- The MaxCompute client has been 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 table creation statement, see [Create a table](#).

2. Use the data transfer tool to upload data.

Example:

```
odpscmd -e "tunnel upload kv.txt wc_in"
```

3. Compile a MaxCompute program and debug it.
 - MaxCompute provides the Eclipse plug-in to help you quickly develop MapReduce programs and debug them on your local machine.
 - You need to create a MaxCompute project in Eclipse, and then compile a MapReduce program in this project. After successful local debugging, upload the compiled program to MaxCompute for base testing.

 **Note** If the Java program requires the use of resources, you must use the `-resources` parameter to specify the resources.

4.11. Compile and run a Graph job

You can submit a Graph job in the same way that you would submit a MapReduce job. This topic provides an example of how to submit a Graph job.

Context

This example uses the SSSP algorithm. The operation procedure is as follows:

Procedure

1. Create input and output tables.

Example:

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

 **Note** For more information about the table creation statement, see [Create a table](#).

2. Use the data transfer tool to upload data.

Example:

```
tunnel u -fd " " sssp.txt sssp_in;
```

3. Compile an SSSP.

 **Note** During the Graph development process, you can locally compile and debug SSSP algorithm examples. You only need to package the SSSP code. You do not need to package the SDK into *odps-graph-example-sssp.jar*.

4. Add JAR resources.

Example:

```
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 resources](#).

5. Run the SSSP.

Example:

```
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. This command is used in the same way as you would use the jar command in MapReduce. The Graph job execution command outputs the job instance ID, execution progress, and result summary. The command output is as follows:

ID = 20170730160742915gl205u3 2017-07-31 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. View job running information

This topic describes how to use the LogView tool to query job running information.

LogView is a tool used to view and debug tasks after a job is submitted to MaxCompute. You can use LogView to view the following content of a job:

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

After a job is submitted to MaxCompute, a link to LogView is generated. You can open the link in a browser to view the job running information.

```
odps@ test_workshop001>select * from result_test;

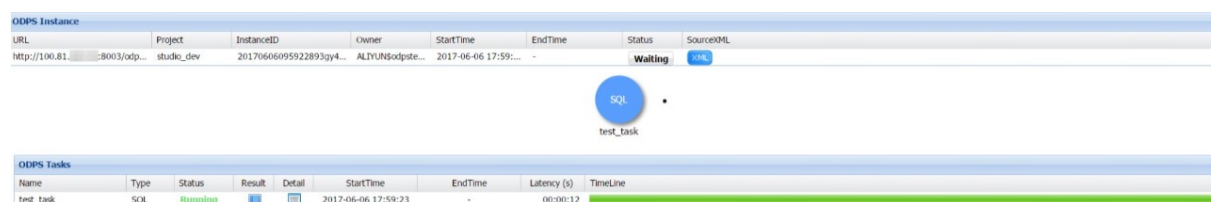
ID = 20190917055240226ga6e5mim
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.com/api&p=test_workshop001&i=20190917055240226ga6e5mim&token=YkxhUzJQ1NIamRWnDBBU3UJPSxPRFBTX09CTzoxMDc5OTI2ODk2OTk5NDIxLDE1NjkzMDOzNjAsnQi0lt7IkFjdGlvbiI6WyJuZHBz01JlYwQiXSwiRmZmZWNOIjoiQWxsY3ciLCJSZXNv3M6b2RwczoqOnByb2plY3RzL3Rlc3Rfd29ya3Nob3AwMDEvaW5zdGFuY2UzLzIwMTkwjI2Z2E2ZTVtaW0iXX1dLCJWZXJzaW9uIjo1MSJ9
Job Queueing...
Summary:

+-----+-----+
| education | num      |
+-----+-----+
```

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

Features

This section describes components on the LogView Web UI.



The LogView page is composed of two parts:

- Instance information
- Task information

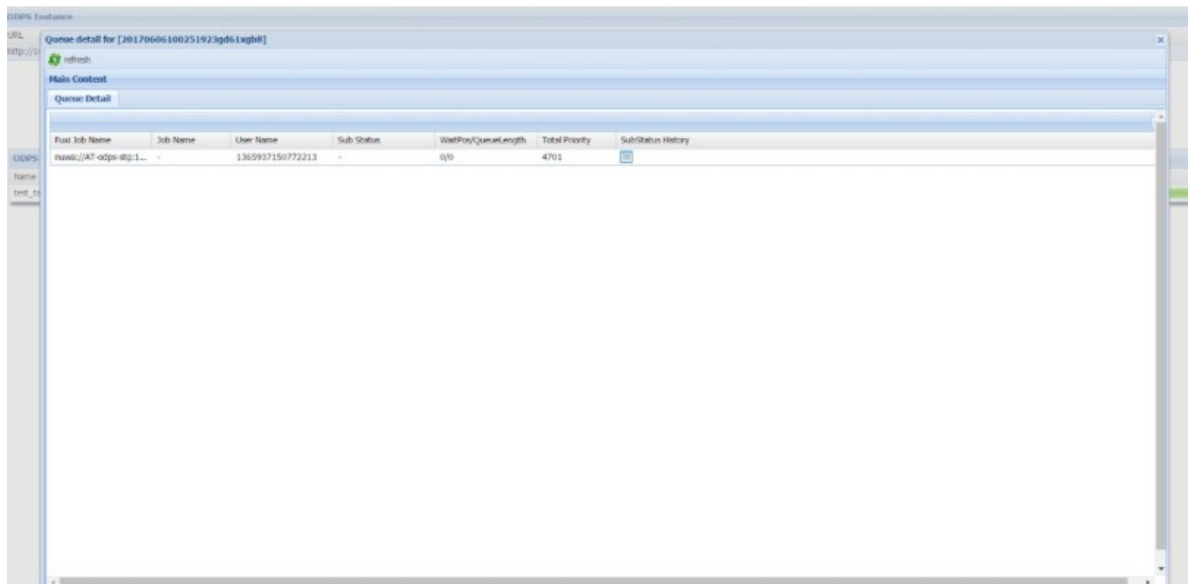
Instance information

The upper pane of the LogView page displays information about the MaxCompute instance corresponding to the SQL tasks that you submit, such as URL, Project, InstanceID, Owner, StartTime, EndTime, and Status.

- If the value of Status is one of the following states, you can click the value to view the queue information:
 - **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 waiting to run in the queue with n-1 other jobs ahead of it.
 - **Running**: The job is running in Job Scheduler.

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

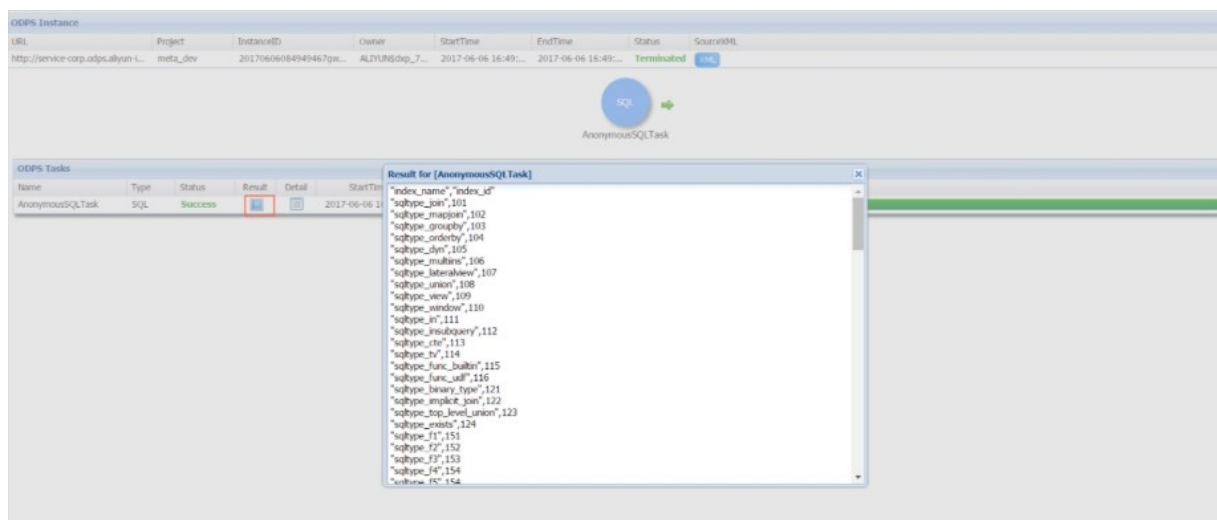
- After you click the value of Status, the following queue information is displayed:
 - **Sub Status**: the current sub-status information.
 - **WaitPos**: the position in the queue. If the value is 0, the job is running. If the value is -, the job has not been submitted to Job Scheduler.
 - **QueueLength**: 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 a detailed status history, such as the status code, status description, start time, and duration of a state. This information is unavailable in some versions.



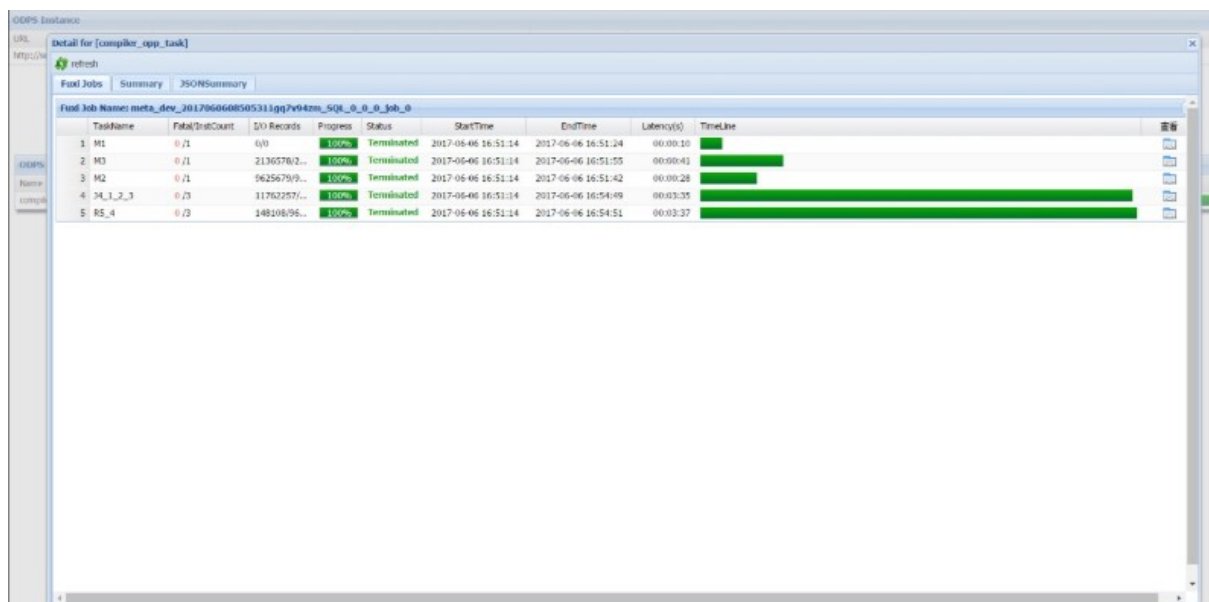
Task information

The lower pane of the LogView page displays the task information corresponding to the instance, such as Name, Type, Status, Result, Detail, StartTime, EndTime, Latency (s), and TimeLine. Like on other pages, latency indicates the total running duration.

Result: You can click the icon in this column to view the task running results after the job is finished. The following figure shows the execution results of a SELECT statement.



Detail: You can click the icon in the **Detail** column to view the task running details regardless of whether the job is running or finished.

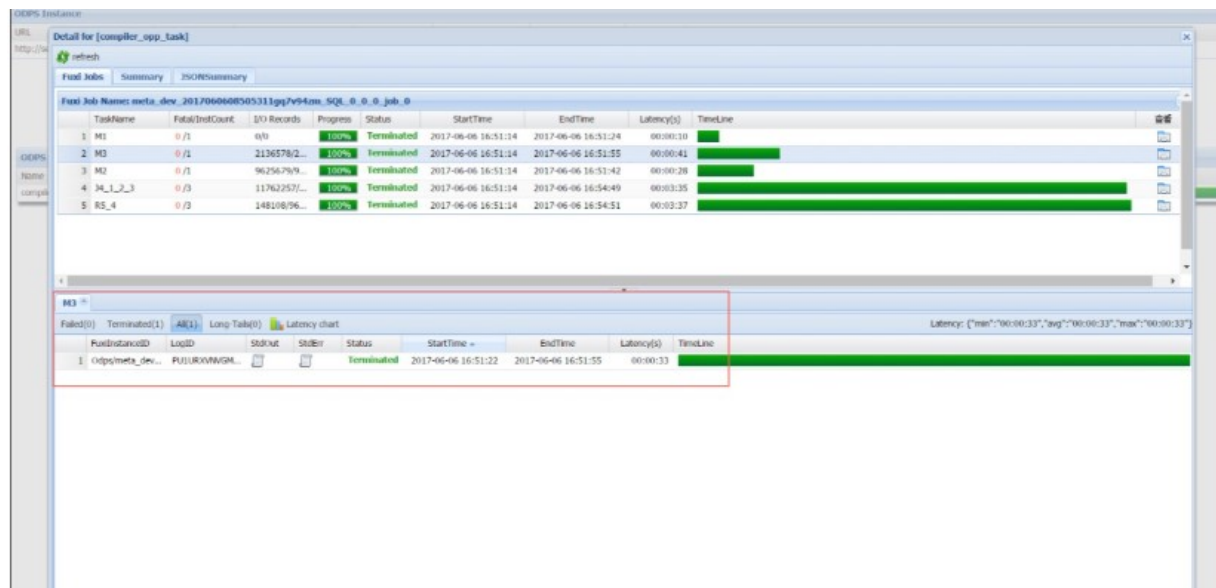


The details about a MaxCompute task are displayed in the detail dialog box. Pay attention to the following points:

- A MaxCompute task consists of one or more Fuxi jobs. For example, if an SQL task that you submit is complex, MaxCompute automatically submits multiple Fuxi jobs to Job Scheduler.
- Each Fuxi job consists of one or more Fuxi tasks. For example, a simple MapReduce task generates two Fuxi tasks: Map Task (M1) and Reduce Task (R2). A complex SQL task may also generate multiple Fuxi tasks.
- The name of a Fuxi task is composed of a letter that indicates the task type and one or more numbers that indicate the task number and its dependencies. For example, in the preceding figure, M1, M2, and M3 are Map tasks, J4_1_2_3 is a Join task that is started after M1, M2, and M3 are finished, and R5_4 is a Reduce task that is started after J4_1_2_3 is finished.
- I/O Records indicates the numbers of input and output records of a Fuxi task.

To view the information about the corresponding instances, you can click the icon in the Show Detail column corresponding to a Fuxi task or double-click the Fuxi task.

Note Each Fuxi task consists of one or more Fuxi instances. As the number of input records of a Fuxi task increases, MaxCompute automatically starts more nodes to help the task process data. Each node corresponds to a Fuxi instance.



In the lower pane of the dialog box, information about the Fuxi instances 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 the standard output information, or click the icon in the StdErr column to view the standard error information.

Note Printed information written in the submitted MaxCompute task is also displayed in the standard output information and standard error information.

Use LogView for troubleshooting

• Tasks with errors

If an error occurs during a task, you can click the icon in the Result column corresponding to the task in the lower pane of the LogView page to view the error information. Alternatively, you can click the icon in the StdErr column corresponding to a Fuxi instance in the detail dialog box to view the error information about the instance.

• Data skew

The long tail of one or more Fuxi instances may slow down the corresponding Fuxi task. The long tail is caused by uneven data distribution in the task. After the task is completed, you can view the running results on the Summary tab of the detail 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 field.

5. Basic concepts and common commands

5.1. Terms

This topic describes the basic terms of MaxCompute.

Project

A basic organizational unit of MaxCompute. Like a database or schema in a traditional database system, a project is the basic unit of multi-user isolation and access control. A user can have permissions on multiple projects. After security 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 my_project
-- Access a project named my_project.
```

 **Note** After you run the preceding command, you are navigated to a project named my_project and gain permissions to manage objects in this project. You are then able to manage objects that belong to this project, regardless of which project you are currently in. The Use Project command is provided by the MaxCompute client. For more information, see [Common commands](#).

Table

A data storage unit of MaxCompute. Logically, a table is a two-dimensional structure consisting of rows and columns, in which each row represents a record and each column represents a field 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. Data in table columns can be any of the data types supported by MaxCompute. Tables are the input and output objects of all MaxCompute computing tasks. You can create and delete tables, or import data to tables.

Partitions of a table can be defined to process data more efficiently. You can specify some fields in the table as partition columns. Partitions within a table are similar to directories within a file system. Each value of a partition 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 the partition that you want to access, MaxCompute only scans the specified partition. This improves processing efficiency and reduces cost. For more information, see [Column and partition operations](#).

Data type

Columns of a MaxCompute table must be of a certain data type. MaxCompute supports the following data types:

Basic data types

Type	New in MaxCompute 2.0?	Constant	Description	Value range
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
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 UTF-8 coded string. The character behaviors of other codes are not defined.	The size of all values in a string column cannot exceed 8 MB.
FLOAT	Yes	None	The 32-bit binary floating point type.	/
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 date and time 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, 99999999999.9999 999BD	The precise numeric type based on the decimal system.	Integer: $-10^{36}+1$ to $10^{36}-1$ Fractional: round to 10^{-18}
VARCHAR	Yes	None	The variable-length type. n specifies the length.	1 to 65535
BINARY	Yes	None	The binary data type.	A single binary column cannot exceed 8 MB.

Type	New in MaxCompute 2.0?	Constant	Description	Value range
TIMESTAMP	Yes	Timestamp '2017-11-11 00:00:00.123456789'	The timestamp type. This type is not time zone specific.	0001-01-01 00:00:00 000000000 to 9999-12-31 23:59:59 999999999

Note that if you want to use the new data types in MaxCompute 2.0, you must first execute the following statement to enable the new data type flag: `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 be NULL.

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

Notice

Note the following points when you use the new data types in MaxCompute 2.0:

- After you execute the `set odps.sql.type.system.odps2=true;` statement, it results in the following major impacts:
 - The semantics of the INT keyword change. INT in an SQL statement indicates a 32-bit integer.
 - The semantics of an integer constant change. Take the `SELECT 1 + a;` statement as an example.
 - If the new data type flag is not enabled, the integer constant is processed as BIGINT. If the length of the constant exceeds the range for a BIGINT value, the integer constant is processed as DOUBLE.
 - If the new data type flag is enabled, the integer constant is 1 of the 32-bit INT type.
 - Possible compatibility issues: The INT type may lead to inconsistencies in function prototypes during subsequent operations. For example, the actions of peripheral tools and subsequent operations might be changed by new type tables generated after data is written to a disk.
 - Implicit type conversion rules change.

If the new data type flag is enabled, some implicit types may not be converted. For example, errors may occur or precision may be reduced when 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. In these cases you can use the CAST function to convert the data type.

Implicit type conversion greatly affects functions and INSERT statements. For example, an INSERT statement can be executed when the new data type flag is disabled, but returns an error when the flag is enabled.

- When the new data type flag is disabled, some operations and built-in functions that use new data types as parameters and response values are ignored. When the new data type flag is enabled, they become valid.
 - Some built-in functions can only be used after the new data type flag is enabled. This includes most functions that use INT type parameters and subsequently suffer from BIGINT overload, such as YEAR, QUARTER, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND, NANOSECOND, DAYOFMONTH, and WEEKOFYEAR. These functions can be implemented by using the DATEPART function.
 - UDF resolution changes.
- The resolution of the BIGINT keyword changes.
- The partition column types change.
 - If the new data type flag is disabled, the partition column type can only be STRING.
 - If the new data type flag is enabled, the partition column type can be STRING, VARCHAR, CHAR, TINYINT, SMALLINT, INT, or BIGINT.
 - If the new data type flag is disabled, partition fields in INSERT operations are processed as STRING.
- The behavior of the LIMIT statement changes.

Take the `SELECT * FROM t1 UNION ALL SELECT * FROM t2 limit 10;` statement as an example.

- If the new data type flag is disabled, it is `SELECT * FROM t1 UNION ALL SELECT * FROM ( SELECT * FROM t2 limit 10) t2;`
- If the new data type flag is enabled, it is `SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2 ) t limit 10;`

Actions of the UNION, INTERSECT, EXCEPT, LIMIT, ORDER BY, DISTRIBUTE BY, SORT BY, and CLUSTER BY statements also change if the new data type flag is enabled.

- The resolution of the IN expression changes.

Take the a in (1, 2, 3) expression as an example.

- If the new data type flag is disabled, all the values in the parentheses () must be of the same type.
 - If the new data type flag is enabled, all the values in the parentheses () are implicitly converted to the same type.
- If the value of a constant is greater than the maximum value of INT but less than the maximum value of BIGINT, it is converted to BIGINT. If the constant is greater than the maximum value of BIGINT, it is converted to DOUBLE. If `odps.sql.type.system.odps2` is not set to true, MaxCompute retains the conversion and notifies you that the INT data is being processed as the BIGINT type. If `odps.sql.type.system.odps2` is set to true, we recommend that you change these types to BIGINT to prevent confusion.
 - VARCHAR constants can be expressed through implicitly converted 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 in milliseconds cannot be displayed. You can add `-dfp` in the Tunnel

command to specify the time display format in milliseconds.

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

Complex data types

Type	Definition	Constructor
Array	array< int >; array< struct< a:int, b:string >>	array(1, 2, 3); array(array(1, 2), array(3, 4))
Map	map< string, string >; map< smallint, array< string>>	map("k1", "v1", "k2", "v2"); map(15, array('a', 'b'), 25, array('x', 'y'))
Struct	struct< x:int, y:int>; struct< field1:bigint, field2:array< int>, field3:map< int, int>>	named_struct('x', 1, 'y', 2); named_struct('field1', 100L, 'field2', array(1, 2), 'field3', map(1, 100, 2, 200))

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

- You can 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 flag.
- You can enable the new data type flag by using DataFrame. For example, use an immediately executed action, such as `persist`, `execute`, or `to_pandas`, by setting the `hints` parameter. The setting in the following figure is 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'})
```

- To enable the new data type flag by using DataFrame and have the setting take effect globally, you need to execute `options.sql.use_odps2_extension = True`.

Resource

A concept used in MaxCompute. To accomplish tasks by using user-defined functions (UDFs) or MapReduce, you must use resources.

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

MapReduce job.

 Note

- There are some limits 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 the examples for resource use and [UDTF instructions](#).

MaxCompute supports the following resource types:

- File
- Table: tables in MaxCompute.
- JAR: compiled JAR packages.
- Archive: compressed files identified by the resource name extension. Supported file types include .zip, .tgz, .tar.gz, .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 implement certain computing or counting functions. If these built-in functions do not meet your requirements, you can use the Java APIs provided by MaxCompute to develop UDFs. UDFs are classified into user-defined scalar functions (UDSFs), user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs).

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

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

Task

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

An execution plan consists of several mutually 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 stages based on the dependencies in the graph (execution plan). A stage has multiple processes, also known as workers. The workers in each stage cooperate to complete computations for the stage. Different workers in a stage process different data, but they all use the same execution logic.

A computing task converts to an instance when it is executed. You can perform operations on this instance, such as obtaining status information and killing the instance.

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

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

Task instance

Some MaxCompute tasks are converted to 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, while 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 and change its status to Canceled.
```

Resource quota

There are two types of resource quota: storage and computing. The storage quota is the upper limit of storage space 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 used to run processes in a project cannot exceed the computing quota.

5.2. Common commands

5.2.1. Introduction

MaxCompute allows you to perform operations on objects, such as projects, tables, resources, and instances. You can use client commands or the SDK to perform operations on these objects.

This topic describes how to run these commands in the MaxCompute client.

 **Note**

- For more information about how to install and configure the client, see [Configure the client](#).
- For more information about the SDK, see [SDK introduction](#).

5.2.2. Project operations

This topic describes common project commands.

Create or delete a project

MaxCompute does not provide commands for creating or deleting projects. You can configure or operate your projects on the console.

Access a project

Command syntax:

```
use <project_name>;
```

Purpose: It is used to access the specified project. After you enter a project, you can directly operate all objects in the project.

 **Note** If the specified project does not exist or you have not been added to the project, the system returns an exception and exits.

Example:

```
odps@ myproject> use my_project;  
-- my_project is a project that you have permission to access.
```

 **Note**

- The preceding command runs in the client.
- All command keywords, project names, table names, and column names in MaxCompute are case-insensitive.
- After you run the command, you can directly access objects in this project. Example:

Run the following command to access the test_src table in the my_project project (assume test_src exists in my_project):

```
odps@ myproject>select * from test_src;
```

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

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

```
odps@ myproject>select * from my_project2.test_src;
```

Data of test_src in my_project2, not my_project, is returned.

View projects

Command syntax:

```
list projects;
```

Purpose: It is used to display all projects in MaxCompute.

Clear objects from a project

Run the following command to view objects in the recycle bin:

```
show recyclebin;
```

Purpose: It is used to list all objects in the project recycle bin.

 **Note** Only the project owner can run this command.

Run the following command to clear all objects from the project recycle bin:

```
purge all;
```

Purpose: It is used to clear all objects from the project recycle bin to release storage space.

 **Note** Only the project owner can run this command.

Run the following command to clear a table:

```
purge table tblname;
```

Purpose: It is used to clear all objects in a specified table from the recycle bin to release the storage space.

 **Note**

- If the specified table exists, the project owner and users who have write permissions on the table can run this command.
- If the table has been deleted using a DROP command, only the project owner can run this command.

5.2.3. Table operations

This topic describes common commands for table operations.

Create a table (CREATE 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];
CREATE TABLE [IF NOT EXISTS] table_name
LIKE existing_table_name;
```

Description: Creates a table.

Example

```
CREATE TABLE IF NOT EXISTS sale_detail( shop_name STRING,
customer_id STRING, total_price DOUBLE)
PARTITIONED BY (sale_date STRING,region STRING);
-- If a table named sale_detail does not exist, a partitioned table with this name is created.
```

 Note

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

Change the owner of a table (CHANGEOWNER)

Syntax

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

Description: Changes the owner of a table.

Example

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

Delete a table (DROP TABLE)

Syntax

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

Description: Deletes a table.

 **Note** If a command without the IF EXISTS option is executed and the table does not exist, an exception is returned. If a command with this option is executed, a success is returned regardless of whether the table exists.

Parameters

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

Example

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

View table information (DESC)

Syntax

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

Description: Returns the information about a specified table. The returned information includes Owner, Project, CreateTime, LastDDLTime, LastModifiedTime, InternalTable (indicates that the object is a table. The value is always YES), Size (table size in bytes), Native Columns (information about non-partition columns, including field, type, and comment), Partition Columns (information about partition columns, including partition name, type, and comment), and Extended Info (information about the external table, such as StorageHandler and Location).

Parameters

table_name: In the first command, it indicates the name of a table or view. In the second command, it indicates the name of an external table.

Example

```
odps@ project_name>DESC sale_detail;
-- Describe a partitioned table.
+-----+
| Owner: ALIYUN$odpsuser@aliyun.com | Project: test_project |
|TableComment: |
+-----+
|CreateTime: 2017-01-01 17:32:13 |
|LastDDLTime: 2017-01-01 17:57:38 |
|LastModifiedTime: 2017-01-01 18:00:00 |
+-----+
|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

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

View partition information (DESC)

Syntax

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

Description: Views the partition information of a partitioned table.

Example

```
odps@ project_name>desc meta.m_security_users partition (ds='20151010');
+-----+
| 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 (SHOW TABLES/SHOW TABLES LIKE)

Syntax

```
SHOW TABLES;
SHOW TABLES like 'chart'
```

Description

- **SHOW TABLES**: shows all tables in the current project.
- **SHOW TABLES like 'chart'**: shows the tables whose name matches chart in the current project. You can use regular expressions in the command to filter tables.

Example

```
odps@ project_name>show tables;
odps@ project_name>show tables like 'ods_brand*';
ALIYUN$odps_user@aliyun.com:table_name
.....
```

 Note

- The preceding commands are executed in the MaxCompute client.
- odps_user@aliyun.com indicates the name of the user who creates the table.
- table_name indicates the table name.

Show partitions (SHOW PARTITIONS)

Syntax

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

Description: Lists all partitions of a table.

Parameters

table_name: the name of the table that you query. An error is returned if the table does not exist or is a non-partitioned table.

Example

```
SHOW PARTITIONS table_name;  
partition_col1=col1_value1/partition_col2=col2_value1  
partition_col1=col1_value2/partition_col2=col2_value2
```

 Note

- The preceding command is executed in the MaxCompute client.
- partition_col1 and partition_col2 indicate the partition columns of the table.
- col1_value1, col2_value1, col1_value2, and col2_value2 indicate the values in the matching columns.

5.2.4. Instance operations

This topic describes common commands for instance operations.

Show instances (SHOW INSTANCES/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: Shows information about instances that you have created.

Parameters

- **startdate To enddate**: a period of time. Information about the instances created within the

specified period is returned. The dates must be in the format of yyyy-mm-dd. This parameter is optional. If it is not specified, information about instances that you have created in the last three days is returned.

- **number**: the number of instances to be returned. 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**: Information about instances executed in the current project is returned. By default, a maximum of 50 instances can be returned. You can run the command only when you have the LIST permission on the current project. To return more than 50 instances, you must use the **-limit number** option. For example, you can run the `show p -all -limit 100` command to return information about 100 instances executed in the current project.
- **project name**: the project name. The account that you use must be a member of the project.

Output fields include StartTime (in seconds), RunTime (in seconds), Status (instance status), instanceID, and the SQL statement corresponding to the instance. The following code provides an example of the command output:

```
odps@ $project_name>show p;
StartTime      RunTime  Status  InstanceID      Owner      Query
2015-04-28 13:57:55  1s      Success  20150428xxxxxxxxxxxxxxxxxxxxx  ALIYUN$xxxxx@aliyun-inc.com
select * from tab_pack_priv limit 20;
...           ...           ...           ...           ...
...           ...           ...           ...           ...
```

An instance can be in any of the following states:

- **Running**
- **Success**
- **Waiting**
- **Failed**: The job failed, but data in the target table is not modified.
- **Suspended**
- **Cancelled**

View the status of an instance (STATUS)

Syntax

```
STATUS <instance_id>;
```

Description: Views the status of a specified instance, which can be Success, Failed, Running, or Canceled.

 **Note** If the instance was not created by you, an exception is returned.

Parameters

instance_id: the unique identifier of an instance. It specifies the instance whose status is queried.

Example

```
odps@ $project_name>status 20131225123xxxxxxxxxxxxxxxxxx;  
Success  
-- Query the status of the instance whose ID is 20131225123xxxxxxxxxxxxxxxxxx. The query result is Success.
```

 **Note** The preceding command is run in the MaxCompute client.

Show running jobs (TOP INSTANCE)

Syntax

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

 **Note** Only the project owner and administrator can use the command.

Description

- **top instance**: obtains information about running jobs that you submit in the current project. The output fields include InstanceID, Owner, Type, StartTime, Progress, Status, Priority, RuntimeUsage (CPU/MEM), TotalUsage (CPU/MEM), and QueueingInfo (POS/LEN).
- **top instance -all**: obtains information about all running jobs in the current project. By default, a maximum of 50 jobs are returned. To return more records, use the -limit number option.

Example

```
odps@ $project_name>top instance;
```

 **Note** The preceding command is run in the MaxCompute client.

Stop an instance (KILL)

Syntax

```
kill <instance_id>;
```

Description: Stops an instance. You can only stop an instance in the Running state. Note that this is an abnormal process. A return from the command only means that the system has received the request. It does not mean that the job has been stopped. Therefore, you must run the STATUS command to view the instance status.

Parameters

instance_id: the unique identifier of an instance. It must be the ID of a running instance. Otherwise, an error is returned.

Example

```
odps@ $project_name>kill 20131225123xxxxxxxxxxxxxxxxxx;  
-- Stop the instance whose ID is 20131225123xxxxxxxxxxxxxxxxxx.
```

 **Note** The preceding command is run in the MaxCompute client.

Describe an instance (DESC INSTANCE)

Syntax

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

Description: Obtains information about the job corresponding to an instance. The obtained information includes the SQL statement, owner, start time, end time, and status.

Parameters

instance_id: the unique identifier of an instance.

Example

```
odps@ $project_name> desc instance 20150715xxxxxxxxxxxxxxxxx;
ID                                     20150715xxxxxxxxxxxxxxxxx
Owner                                ALIYUN$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;
-- Query information about the job corresponding to the instance whose ID is 20150715xxxxxx
xxxxxxxxxx.
```

 **Note** The preceding command is run in the MaxCompute client.

Obtain task operation log information (WAIT)

Syntax

```
wait instance_id;
```

Description: Obtains task operation log information based on an instance ID. The returned information includes a link to LogView. You can view log details by using the link.

Parameters

instance_id: the unique identifier of an instance.

Example

```

wait 20170925161122379gxxxxxx;
ID = 20170925161122379g3xxxxxx
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=aliam&i=2017
0925161122379g3xxxxxx&token=XXXXXXvbiI6IjEifQ==
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)
-- Query the task operation logs of the instance whose ID is 20170925161122379gxxxxxx.

```

 **Note** The preceding command is run in the MaxCompute client.

5.2.5. Resource operations

This topic describes common commands for resource operations.

Add a resource (ADD FILE/ARCHIVE/TABLE/JAR/PY)

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: Adds resources to MaxCompute.

The following table lists parameters in this command.

Parameters

Parameter	Description
<code>file/archive/table/jar/py</code>	Indicates the resource type. For more information about resource types, see Resource in Terms .
<code>local_file</code>	Indicates the path of a local file. The file name is used as the resource name, which uniquely identifies the resource.
<code>table_name</code>	Indicates the name of a table in MaxCompute.
<code>[PARTITION (spec)]</code>	If the resource that is added is a partitioned table, MaxCompute only takes a partition as a resource, as opposed to the whole partitioned table.
<code>alias</code>	Indicates the resource name. If this parameter is not specified, the file name is used as the resource name. JAR and Py resources do not support this parameter.
<code>[comment 'cmt']</code>	Indicates a comment on the resource.
<code>[-f]</code>	If a resource with the same name exists, this operation overwrites the existing resource. If this option is not specified and a resource with the same name exists, the operation fails.

Example

```
odps@ odps_public_dev>add table sale_detail partition (ds='20170602') as sale.res comment '
sale detail on 201706 02' -f;
OK: Resource 'sale.res' have been updated.
-- Add a table resource with alias sale.res to MaxCompute.
```

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

Delete a resource (DROP RESOURCE)

Syntax

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

Description: Deletes an existing resource.

Parameters

resource_name: the name of the resource to be deleted. It is specified when the resource is created.

View the resource list (LIST RESOURCES)

Syntax

```
LIST RESOURCES;
```

Description: Lists all resources in the current project.

Example

```
odps@ $project_name>list resources;
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 (GET RESOURCE)

Syntax

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

Description: Downloads a resource from MaxCompute to a local device. You can download file, JAR, archive, and Py resources, but not table resources.

Parameters

- **resource_name**: the name of the resource to be downloaded.
- **path**: the local path to save the resource.

Example

```
odps@ $project_name>get resource odps-udf-examples.jar d:\;
OK
```

5.2.6. Function operations

This topic describes common commands for function operations.

Register a function (CREATE FUNCTION)

Syntax

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

The following table lists parameters in this command.

Parameters

Parameter	Description
function_name	Indicates the UDF name. This name is used in SQL statements to reference this function.

Parameter	Description
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 UDF implementation class name, which are separated with periods (.). For a Python UDF, package_to_class indicates the Python script name and class name, which are separated with periods (.). This name must be enclosed in a pair of single quotation marks (').
resource_list	Indicates the list of resources that the UDF uses. The list must include the resource where the UDF code is located. Ensure that the resource is 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.

Example

- 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 my_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 the project owner has the permission to use UDFs to overwrite built-in functions. If you use a UDF that overwrites a built-in function, warning information is included in Summary after the SQL statement is executed.

Delete a function (DROP FUNCTION)

Syntax

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

Parameters

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

Example

```
DROP FUNCTION test_lower;
```

View the function list (LIST FUNCTIONS)

Syntax

```
LIST FUNCTIONS;  
-- View all UDFs in the current project.  
LIST FUNCTIONS -p project_name;  
-- View all UDFs in a specified project.
```

5.2.7. Time zone configuration operations

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

Time zone configurations

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

- **Session level**: Submit the set odps.sql.timezone=<timezoneid>; statement along with a computing statement for execution. Example:

```
set odps.sql.timezone=Asia/Tokyo;  
-- Set the time zone to Asia/Tokyo.  
select getdate();  
-- Query the current time zone.  
output:  
+-----+  
| _c0      |  
+-----+  
| 2018-10-30 23:49:50 |  
+-----+
```

- **Project level**: Execute the setProject odps.sql.timezone=<timezoneid>; statement. Only the project owner has the permission to execute this statement.



Notice After the time zone is set at the project level, it is used in all computing tasks that are time related. The data of existing tasks are also affected. Therefore, set the time zone only when it is absolutely necessary. We recommend that you only set time zones for new projects.

Limits and precautions

- SQL built-in date functions, UDFs, UDTs, UDJs, and the SELECT TRANSFORM statement all support time zone configuration at the project level.
- 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 to STRING.
- After you configure the time zone, the output time may be different from the real time when 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.
- New versions of MaxCompute, Java SDK, and the related console are provided to ensure that the DATETIME data stored in MaxCompute is correct in multiple time zones. The target Java SDK and console versions have the -oversea suffix. The upgrade may affect the display of the DATETIME data that is before January 1, 1928 in MaxCompute.
- If the local time zone is not UTC+8, we recommend that you upgrade the Java SDK and console to ensure that the SQL-based computing results and data transferred using Tunnel after January 1, 1900 are accurate and consistent. After the upgrade, for the DATETIME data before January 1, 1900, the SQL-based computing results and data transferred 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 upgrade, the time in the new version is 352 seconds earlier.
- If you do not upgrade the Java SDK and console to the version with the -oversea suffix, the SQL-based computing results and data transferred using Tunnel will 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 time zone of the upgraded Java SDK or console version 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 set the time zone on the client to ensure the time consistency between the client and the server.
- MapReduce supports time zone configuration.
- Spark supports time zone configuration.
 - If tasks are submitted to the 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 the spark.driver.extraJavaOptions -Duser.timezone=America/Los_Angeles statement. The timezone parameter in the preceding statement indicates the time zone to be used.
- PAI supports time zone configuration.
- Graph supports time zone configuration.

5.2.8. Tunnel operations

This topic describes common commands for Tunnel operations.

Features

- **UPLOAD:** uploads files or directories (level-1 directories). Data can be uploaded only to a table or a partition of a table. For a partitioned table, you must specify the partition where data is upload.

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

- **DOWNLOAD:** downloads a table or partition to a single file. For a partitioned table, specify the partition to be downloaded.

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

- **RESUME:** resumes the transfer of files or directories after a network or Tunnel service error occurs. Dship supports resumable data transfer.

```
tunnel resume;
```

- **SHOW:** displays historical task information.

```
tunnel show history -n 5
tunnel show log
```

- **PURGE:** clears the session directory. Sessions within the past three days are purged by default.

```
tunnel purge 5
```

Use of Tunnel commands

Tunnel commands allows you to obtain help information by using the Help subcommand on the client. Each command and option support short command format:

```
odps@ project_name>tunnel help;
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.
```

The following table describes parameters in this command.

Parameters

Parameter	Description
upload	Uploads data to MaxCompute tables.
download	Downloads data from MaxCompute tables.
resume	Resumes data upload if a failure occurs. Currently, data download cannot be resumed. Each data upload or download operation is called a session. To resume a session, you must specify the session ID in the RESUME command.
show	Displays historical running information.

Parameter	Description
purge	Clears the session directory.
help	Provides Tunnel help information.

UPLOAD

Imports data from a local file to a MaxCompute table in append mode. Subcommand syntax:

```
usage: tunnel upload [options] <path> <[project.]table[/partition]>
       upload data from local file
  -bs, -block-size <ARG>          block size in MiB, default 100
  -c, -charset <ARG>              specify file charset, default ignore.
                                   set ignore to download raw data
  -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)
  -rd, -record-delimiter <ARG>    specify record delimiter, support
                                   unicode, eg \u0001. default "\n"
  -s, -scan <ARG>                specify scan file
                                   action(true|false|only), default true
  -sd, -session-dir <ARG>         set session dir, default /D:/console/plugins/dship/
  -te, -tunnel_endpoint <ARG>     tunnel endpoint
  -threads <ARG>                  number of threads, default 1
  -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 parameters in this command.

Parameters

Parameter	Description
-bs, -block-size	Specifies the size of each data block uploaded using Tunnel. Default value: 100 MiB (1 MiB = 1,024 × 1,024 bytes).
-c, -charset	Specifies the encoding format of local data files. The default value is UTF-8 without timing. The source data is downloaded by default.
-cp, -compress	Specifies whether to compress the local file before uploading it to reduce traffic. Compression is enabled by default.

Parameter	Description
-dbr	Specifies whether to ignore dirty data, such as additional columns, missing columns, and unmatched types of column data. If the value is true, all data that does not comply with table definitions is ignored. If the value is false, an error is returned when dirty data is found, and raw data in the destination table is not contaminated.
-dfp	Specifies the format of DATETIME data. The default format is yyyy-MM-dd HH:mm:ss.
-fd	Specifies the column delimiter used in the local data file. The default delimiter is comma (,).
-h	Specifies whether the data file has a table header. If the value is true, Dship skips the table header and starts uploading data from the second row.
-mbr,-max-bad-records	If more than 1,000 rows of dirty data are uploaded, the upload operation is terminated by default. This parameter allows you to adjust the maximum allowable volume of dirty data.
-ni	Specifies the NULL data identifier. The default value is an empty string ("").
-rd	Specifies the row delimiter in the local data file. The default value is \n in a Linux system or \r\n in a Windows system.
-s	Specifies whether to scan the local data file. The default value is false. If the value is true, data is scanned first, and is imported if the format is correct. If the value is false, data is imported without scanning. If the value is only, data is scanned but not imported.
-sd,-session-dir	Specifies the path of the session directory. The default value is <i>/D:/console/plugins/dship/lib/...</i>
-te	Specifies the endpoint of Tunnel.
-tz	Specifies the time zone. The default value is Asia/Shanghai.

Example

Create a 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 a partition:

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

Prepare the data file data.txt with the following content:

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

The third row of this file does not comply with the definitions of the sale_detail table. sale_detail defines three columns, but the third row of this file contains only two columns. Run the following command to import the data:

```
odps@ project_name>tunnel u d:\data.txt      sale_detail/sale_date=201705,region=hangzhou -s
false 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 deli
miter
```

The data import fails because of the dirty data in data.txt. The session ID and error message are displayed. You can run the following command to verify the upload result:

```
odps@ odptest_ay52c_ay52> select * from sale_detail where sale_date='201705'; ID = 2017061
0084135370gyvc61z5
+-----+-----+-----+-----+-----+
shop_name | customer_id | total_price | sale_date | region |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

DOWNLOAD

Subcommand syntax

```

odps@ project_name>tunnel help download
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
-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 "\n"
-sd,-session-dir <ARG>       set session dir, default /D:/console/plugins/dsh
                             ip/
-te,-tunnel_endpoint <ARG>   tunnel endpoint
-threads <ARG>               number of threads, default 1
-tz,-time-zone <ARG>         time zone, default local timezone:Asia/Shanghai
Example:
tunnel download test_project.test_table/p1="b1",p2="b2" log.txt

```

The following table describes parameters in this command.

Parameters

Parameter	Description
-fd	Specifies the column delimiter used in the local data file. The default delimiter is a comma (,).
-rd	Specifies the row delimiter used in the local data file. The default delimiter is \r\n.
-dfp	Specifies the format of DATETIME data. The default format is yyyy-MM-dd HH:mm:ss.
-ni	Specifies the NULL data identifier. The default value is an empty string ("").
-c	Specifies the encoding format of local data files. The default value is UTF-8.

Example

Download data to result.txt:

```
$ ./tunnel download sale_detail/sale_date=201705,region=hangzhou result.txt; Download session: 201706101658245283870a002ed0b9
Total records: 2
2017-06-10 16:58:24 download records: 2
2017-06-10 16:58:24 file size: 30 bytes
OK
```

Open the result.txt file and verify that its content is as follows:

```
shop9,97,100.0
shop10,10,200.0
```

RESUME

Resumes the execution of historical operations. Only data upload can be resumed. Subcommand syntax:

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

Example

Change the content of the data.txt file to the following items:

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

Resume the data upload:

```
odps@ project_name>tunnel resume 201706101639224880870a002ec60c --force;
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
```

201706101639224880870a002ec60c is the ID of the session that failed to be uploaded. You can run the following command to verify the upload result:

```
odps@ project_name>select * from sale_detail where sale_date='201705';
ID = 20170610084801405g0a741z5
+-----+-----+-----+-----+-----+
shop_name | customer_id | total_price | sale_date | region |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

SHOW

Displays historical records. Subcommand syntax:

```
usage: tunnel show history [options]
```

Example

```
odps@ project_name>tunnel show history;
usage: tunnel show history [options]
show session information
-n, -number <ARG>    lines
Example:
tunnel show history -n 5
tunnel show log
```

PURGE

Clears the session directory. Sessions within the last three days are purged by default. Subcommand syntax:

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

5.2.9. Other operations

This topic describes common commands for other operations.

ALIAS

The ALIAS command is used to create an alias for a resource. You can read different resources from the MaxCompute MapReduce reference manual or UDF code by creating the same alias for these resources.

Syntax

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

Description: Creates an alias for a resource.

Parameters

- **alias**: the alias of a resource.

- **real**: the original name of a resource.

Example

```
ADD TABLE src_part PARTITION (ds='20171208') AS res_20171208;
ADD TABLE src_part PARTITION (ds='20171209') AS res_20171209;
-- Add resources res_20171208 and res_20171209.
ALIAS resName=res_20171208;
jar -resources resName -libjars work.jar -classpath ./work.jar com.company.MainClass args .
..;
-- Set the alias of resource res_20171208 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 resource res_20171209 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: Configures built-in or user-defined system variables of MaxCompute to affect MaxCompute behavior.

Parameters

- **KEY**: the name of the attribute to be set.
- **VALUE**: the value of the attribute.

MaxCompute SQL and the latest MapReduce version support the following SET commands:

```

set odps.stage.mapper.mem=
-- Set the memory size of each map worker. Unit: MB. Default value: 1024.
set odps.stage.reducer.mem=
-- Set the memory size of each reduce worker. Unit: MB. Default value: 1024.
set odps.stage.joiner.mem=
-- Set the memory size of each join worker. Unit: MB. Default value: 1024.
set odps.stage.mem =
-- Set the memory size of all workers of a specified MaxCompute task. This command has a lower priority than the preceding commands. Unit: MB. This parameter is not specified by default.
set odps.stage.mapper.split.size=
-- Set the input data volume of each map worker (size of each slice in the input file) to indirectly control the number of workers in each map stage. Unit: MB. Default value: 256.
set odps.stage.reducer.num=
-- Set the number of workers in each reduce stage. This parameter is not specified by default.
set odps.stage.joiner.num=
-- Set the number of workers in each join stage. This parameter is not specified by default.
set odps.stage.num=
-- Set the number of concurrent workers in all stages of a specified MaxCompute task. This command has a lower priority than the preceding three commands. This parameter is not specified by default.

```

The earlier MapReduce versions of MaxCompute support the following SET commands:

```

set odps.mapred.map.memory=
-- Set the memory size of each map worker. Unit: MB. Default value: 1024.
set odps.mapred.reduce.memory=
-- Set the memory size of each reduce worker. Unit: MB. Default value: 1024.
set odps.mapred.map.split.size=
-- Set the input data volume of each map worker (size of each slice in the input file) to indirectly control the number of workers in each map stage. Unit: MB. Default value: 256.
set odps.mapred.reduce.tasks=
-- Set the number of workers in each reduce stage. This parameter is not specified by default.

```

Example

- 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 input data volume of each map worker to 256 MB.

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

SETPROJECT

Syntax

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

Description: Sets project attributes. If `< KEY >=< VALUE >` is not specified, the current attribute configurations of the project are displayed.

The following table describes project attributes.

Project attributes

Attribute	Permission owner	Description	Value range
<code>odps.table.drop.ignorenonexistent</code>	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
<code>odps.instance.priority.autoadjust</code>	Project owner	Indicates whether to automatically prioritize smaller tasks. If the value is true, this function is enabled.	true and false
<code>odps.instance.priority.level</code>	Project owner	Indicates the priority of a task in a project.	1 to 3  Note The value 1 indicates the highest priority.
<code>odps.security.ip.whitelist</code>	Project owner	Indicates an IP address whitelist for the project.	List of IP addresses separated with commas (,).
<code>odps.table.lifecycle</code>	Project owner	optional: The lifecycle clause is optional in a table creation statement. If no lifecycle is set for a table, the table does not expire. mandatory: The lifecycle clause is mandatory. inherit: If no lifecycle is set for a table, the value of <code>odps.table.lifecycle.value</code> is the lifecycle of this table.	optional, mandatory, inherit

Attribute	Permission owner	Description	Value range
<code>odps.table.lifecycle.value</code>	Project owner	Indicates the default lifecycle.	1 to 37231  Note The default value is 37231.
<code>odps.instance.remain.days</code>	Project owner	Indicates how long the information about an instance is retained. Unit: days.	3 to 30
<code>odps.task.sql.outerjoin.ppd</code>	Project owner	Indicates whether the filter conditions in FULL OUTER JOIN are pushed down.	true and false
<code>odps.function.strict mode</code>	Project owner	Indicates whether to return NULL (false) or report an error (true) when built-in functions have dirty data.	true and false
<code>odps.task.sql.write.str2null</code>	Project owner	Indicates whether to consider empty strings as NULL (true).	true and false

EXPORT project meta

Syntax

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

Description: Exports the meta of a project to a local file. Meta is represented by statements acceptable by `odpscmd`. The exported meta file can be used to recreate a project.

SHOW FLAGS

Syntax

```
show flags;
```

Description: Displays parameters configured by using the SET command.

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

COST SQL

Syntax

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

Description: Estimates the measurement information of an SQL statement, including the size of the input data, number of UDFs, and SQL complexity level.

Example

```
odps@ $odps_project >cost sql select distinct project_name, user_name from meta.m_security_
users distribute by project_name sort by project_name;
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 large volumes of data (terabytes) need to be processed, but do not have high real-time requirements. In such scenarios, it takes a relatively long time to prepare and submit each job. MaxCompute SQL is not well-suited for businesses that require to process thousands of transactions per second. MaxCompute SQL online computing provides near real-time (NRT) processing capabilities.

MaxCompute SQL uses the syntax that is similar to SQL syntax. It can be considered as a subset of standard SQL. However, MaxCompute SQL is not equivalent to a database. It does not have common database characteristics, such as transactions, primary key constraints, and indexes. The maximum length of SQL statements currently supported by MaxCompute is 2 MB.

6.1.2. Reserved words

Keywords of SQL statements are reserved words in MaxCompute. Do not use reserved words to name tables, columns, or partitions. Otherwise, an error is returned. Reserved words are case-insensitive.

Common reserved words are listed as follows. 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

Partition columns provide many benefits, such as higher SQL operating efficiency and lower costs. However, too many partitions can cause problems. Using partition columns as filtering conditions in WHERE clauses of SELECT statements can bring greater benefits. Some SQL partition statements run inefficiently. For example, a statement fails when a large volume of data (more than 2,048 MB) is generated in dynamic partitions in a single MaxCompute instance.

It is easy to underestimate the number of partitions generated when multi-level partitions are used. When a huge number of partitions are generated, you must evaluate the original data to determine if there are excessive partitions.

You can create up to six levels of partitions. For some MaxCompute commands, the syntax differs between partitioned and non-partitioned tables. For more information, see [DDL statements](#) and [DML statements](#).

For more information about the table creation statement, see [Create a table](#).

6.1.4. Type conversion

6.1.4.1. Explicit type conversion

Explicit conversion uses CAST to convert a value type to another one. This topic describes explicit type conversion.

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

Explicit type conversion

From/To	Bigint	Double	String	Datetime	Boolean	Decimal
Bigint	-	Y	Y	N	N	Y
Double	Y	-	Y	N	N	Y
String	Y	Y	-	Y	N	Y
Datetime	N	N	Y	-	N	N
Boolean	N	N	N	N	-	N
Decimal	Y	Y	Y	N	N	-

Y indicates that the type can be converted. N indicates that the type cannot be converted.

Note

- When double type values are converted to bigint, the fractional is truncated. For example, `cast(1.6 as bigint) = 1`.
- When a string that meets double type requirements is converted to bigint, the string is first converted to the double type before it is converted to the bigint type. Hence, the fractional is truncated. For example, `cast("1.6" as bigint) = 1`.
- When a string that meets bigint type requirements is converted to the double type, one decimal is retained. For example, `cast("1" as double) = 1.0`.
- To convert a constant string to the decimal type, enclose the constant string within a pair of quotation marks. If the value is not enclosed in quotation marks, it is treated as a double type value. For example, `cast("1.234567890123456789" as decimal)`.
- Unsupported explicit type conversion operations cause an exception.
- If a conversion fails during execution, the system returns an error and exits.
- The datetime data conversion uses the default format `yyyy-mm-dd hh:mi:ss`. For more information, see [Convert data between string and datetime types](#).
- Some types cannot be explicitly converted, but can be converted using built-in SQL functions. For example, the `to_char` function can be used to convert boolean type values to the string type. For more information, see [TO_CHAR](#). The `to_date` function can be used to convert string type values to the datetime type. For more information, see [TO_DATE](#).
- For more information about CAST, see [CAST](#).
- When the values of the decimal type are out of 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

Implicit type conversion is an automatic type conversion performed by MaxCompute based on the context and a predefined set of rules. This topic describes the rules of implicit type conversion.

The following table lists implicit type conversion rules supported by MaxCompute.

Implicit type 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

From/To	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	FLOAT
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 type 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
TIMESTAMP	F	F	T	T	T	F
BINARY	F	F	F	F	F	T

T indicates that the type conversion can be performed, while F indicates that the type conversion cannot be performed.

Note

- An unsupported implicit type conversion will cause an exception.
- If the conversion fails, an error is returned.
- Implicit type conversion is automatically performed by MaxCompute based on context. If the types do not match, we recommend that you perform explicit type conversion using cast.
- The rules of implicit type conversion are applied to different specific scopes. In certain scenarios, only part of the rules will take effect. For more information, see the scope of implicit type conversions.

Implicit type conversion with relational operators

Relational operators include equal to (=), not equal to (<>), less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), IS NULL, IS NOT NULL, LIKE, RLIKE, and IN. The implicit conversion rules of LIKE, RLIKE, and IN are different from those of the other relational operators. These three operators are described in a separate section. The rules described in this section do not apply to these three operators. The following table lists implicit conversion rules when different types of data are involved in relational calculations.

Implicit type conversion with relational operators

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

Note

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

Implicit conversion with special relational operators

Special relational operators are LIKE, RLIKE, and IN.

LIKE and RLIKE are used as follows:

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

Note the following points for the two relational operators in implicit type conversion:

- The source and pattern parameters of LIKE and RLIKE must be of the string type.
- Other types are not supported by this operation and cannot be implicitly converted to the STRING type.
- If the value of source or pattern is NULL, the operation returns NULL.

IN is used as follows:

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

The implicit conversion rules of IN are as follows:

- The data types in the value list specified by IN must be consistent.
- If keys and values are compared, the BIGINT, DOUBLE, and STRING types compared are converted to DOUBLE, whereas the DATETIME and STRING types compared are converted to DATETIME. Conversion between other types is not allowed.

Note the following points for the IN operator:

The memory used by the compiler increases with the number of parameters used by the IN operation. An IN operation with 5,000 parameters consumes 17 GB of memory with the GCC compiler. We recommend that you limit the number of parameters to around 1,024. In this case, memory consumption will peak at 1 GB and compilation will only take 39 seconds.

Implicit type conversion with arithmetic operators

Arithmetic operators include plus (+), minus (-), multiplier (*), divider (/), and percent (%). The implicit conversion rules are as follows:

- Only the STRING, BIGINT, DECIMAL, and DOUBLE types can be used in arithmetic operations.
- Before an arithmetic operation, STRING values are implicitly converted to DOUBLE values.
- When an arithmetic operation involves values of both the BIGINT and DOUBLE types, BIGINT values are implicitly converted to DOUBLE values.
- The DATETIME and BOOLEAN types cannot be used in arithmetic operations.

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

Implicit conversion with logical operators

Logical operators include AND, OR, and NOT. The implicit conversion rules are as follows:

- Only the BOOLEAN type can be used in logic operations.
- The other types are not supported by logical operations or implicit type 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 system functions, which can be used to calculate one or more columns of any row and output any type of data.

The implicit conversion rules are as follows:

- In a call of a function, if the data type of an input parameter is not consistent with the data type defined in the function, the data type of the input parameter is converted to the function-defined

data type.

- The parameters of each built-in SQL function on MaxCompute can have different requirements for implicit type conversion. For more information, see [Built-in functions](#).

6.1.4.4. CASE WHEN

This topic describes the implicit conversion rules of CASE WHEN.

The implicit conversion rules of case when are as follows:

- If the returned data types are only bigint and double, they are converted to the double type.
- If data of the string type is also returned, all data types are converted to string. If a data type cannot be converted to string (for example, boolean), an error is returned.
- Conversion between other types is not allowed.

6.1.4.5. Partition column

MaxCompute SQL supports partitioned tables. For the definition of partitioned tables, see [DDL statements](#) and [DML statements](#). MaxCompute supports partitions of the following types: tinyint, smallint, int, bigint, varchar, and string.

6.1.4.6. UNION ALL

The data type, number of column, and column names involved in UNION ALL operation must all be consistent. Otherwise, an error is returned.

6.1.4.7. Conversion between string and datetime types

MaxCompute supports conversion between string and datetime types.

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

Value ranges of units

Unit	String (case-insensitive)	Value range
Year	yyyy	0001-9999
Month	mm	01-12
Day	dd	01-28,29,30,31
Hour	hh	00-23
Minute	mi	00-59
Second	ss	00-59
ms	ff3	00-999

Note

- Leading zeros cannot be omitted. For example, 2017-1-9 12:12:12 is an invalid string and cannot be converted into datetime. It must be written as 2017-01-09 12:12:12.
- Only strings that meet the preceding format requirements can be converted into datetime. For example, `cast("2017-12-31 02:34:34" as datetime)` converts the "2017-12-31 02:34:34" string into datetime. Similarly, when datetime is converted into strings, the default conversion format is yyyy-mm-dd hh:mi:ss. If you attempt to convert the following examples (or similar strings), the operation will fail and cause an exception.

```
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, which converts a string type that does not meet the datetime format into 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 not equal 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 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.

Operator	Description
A LIKE B	If A or B is NULL, NULL is returned. A is a string and B is the pattern to be matched. 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 ordinary characters, use backslashes to escape them: \% and _. 'aaa'like 'a ' = TRUE'aaa' like'a%' = TRUE'aaa'like 'aab' = FALSE'a%b'like 'a\b' = TRUE'axb'like 'a\b' = FALSE
A RLIKE B	If A or B is NULL, NULL is returned. A is a string and B is a string constant regular expression. If A matches B, TRUE is returned. Otherwise, FALSE is returned. If B is NULL, 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, that is, 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.

Double type values have variable precision. We recommend that you do not use the equal sign (=) to compare two double type values. You can subtract between two values of the double type, and then take the absolute value of the result for comparison. When the absolute value is negligible, the two values of the double type are considered equal. For example:

```
abs(0.9999999999 - 1.0000000000) < 0.0000000001
-- 0.9999999999 and 1.0000000000 have 10 decimal digits, while 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](#).
- 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, the result is of the double type.
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

- Only values of the string, bigint, double, and decimal types can be used in arithmetic operations. Values of the datetime and boolean types are not allowed in these operations.
- Before the operation, values of the string type are converted to the double type by implicit type conversion.
- When values of the bigint and double types are involved in an operation, values of the bigint type are converted to the double type by implicit type conversion first. The returned result is a value of the double type.
- When both A and B are of the bigint type, the returned result of A / B is a value of the double type. The returned results of the other arithmetic operations are values of the bigint type.

6.2.3. Bitwise operators

This topic describes bitwise operators in MaxCompute SQL operators.

Bitwise operators

Operator	Description
A & B	Returns the bitwise AND result of A and B. For example, 1 & 2 returns 0 and 1 & 3 returns 1. The bitwise AND result of NULL in combination with another value is always NULL. A and B must be of the bigint type.
A B	Returns the bitwise OR result of A and B. For example, 1 2 returns 3 and 1 3 returns 3. The bitwise OR result of NULL in combination with another value is always NULL. A and B must be of the bigint type.

 **Notice** Bitwise operators only support bigint type data and do not support implicit type conversion.

6.2.4. Logical operators

This topic describes logical operators in MaxCompute SQL operators.

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 expression A is NULL, NULL is returned.
	If expression A is TRUE, FALSE is returned.
	If expression A is FALSE, TRUE is returned.

 **Note** Only data of the boolean type can be involved in logic operations. These operations do not support implicit type conversion.

6.3. DDL statements

6.3.1. Table operations

6.3.1.1. Create a table (CREATE TABLE)

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

Syntax

```
create 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], ...)]
[STORED AS AliOrc]-- Specify the storage format of the table. You can specify AliORC only f
or newly created internal tables.
[lifecycle days]
[as select_statement]
create table [if not exists] table_name like existing_table_name
```

? Note

- Table and column names are not case-sensitive.
- If you do not specify the IF NOT EXISTS option and another table with the same name exists, an error is returned. If you specify this option, a message that indicates the operation succeeded is returned. The message is returned regardless of whether an existing 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 you want to create. In addition, the metadata of the existing table does not change.
- A table can contain a maximum of 1,200 column definitions.
- Supported data types are BIGINT, DOUBLE, BOOLEAN, DATETIME, DECIMAL, STRING, ARRAY <T>, and MAP <T1, T2>.

? Note If you need to use the following newly supported data types: TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, you must add the `set odps.sql.type.system.odps2=true;` flag before the CREATE TABLE statement. Then, commit them for execution.

- MaxCompute allows you to specify the default value of a column by using DEFAULT value. If the value of a column is not specified in an INSERT operation, the default value is used for this column.
- A table or column name cannot contain special characters. It can contain only lowercase letters, uppercase letters, digits, or underscores (_). A name must start with a letter and can be up to 128 bytes in length.
- The partitioned by option specifies the partition field. The value can only be a string. The name of a partition key column cannot contain double-byte characters. It must start with a letter, either in lowercase or uppercase, followed by letters or digits. The name can be up to 128 bytes in length. The name can contain the following special characters: ! _ : \$. # @ and spaces. Other characters, such as \t, \n, and /, are considered undefined characters. After you use partition fields to define partitions for a table, a full table scan is no longer triggered when you add partitions, update partition data, or read partition data. This improves processing efficiency.
- A comment is a valid string that can be up to 1,024 bytes in length.
- The lifecycle option indicates the lifecycle of the table in days. The CREATE TABLE LIKE statement does not replicate the lifecycle attribute from the source table.
- Theoretically, a source table can have up to six levels of partitions. Use as few partitions as possible to avoid extreme table expansion of storage.
- You can configure the maximum number of table partitions for a project. The default maximum number is 60,000.
- STORED AS specifies 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 extra storage space and improves read performance by more than 20%.

Examples

The following example describes how to create a table named `sale_detail` to store sales records. The `sale_date` and `region` columns of the table are used as partition key columns.

```
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.
```

Use the following `create table...as select...` statement to create a table and replicate data to it:

```
create table sale_detail_ctas1 as select * from sale_detail;
```

Note If the `sale_detail` table contains data, all the data is replicated to `sale_detail_ctas1`. The `sale_detail` table is a partitioned table. However, the table created by the `create table...as select...` statement does not replicate the partition attribute of `sale_detail`. Partition key columns in `sale_detail` become standard columns in `sale_detail_ctas1`. Therefore, `sale_detail_ctas1` is a non-partitioned table that has five columns.

In the `create table...as select...` statement, if you use constants as column values in the `SELECT` clause, we recommend that you specify column aliases:

```
create table sale_detail_ctas2 as select shop_name,
customer_id, total_price,
'2017' as sale_date,
'China' as region from sale_detail;
```

Note

If you do not specify column aliases, the fourth and fifth columns of `sale_detail_ctas3` created in the following example are automatically named `_c3` and `_c4`.

```
create table sale_detail_ctas3 as select shop_name,
customer_id, total_price, '2017',
'China'
from sale_detail;
```

In this case, to reference `sale_detail_ctas3` again, you must enclose `_c3` and `_c4` in two pairs of grave accents (`'`). If you execute the `select c3, _c4 from sale_detail_ctas3` statement, an error is returned. The column name in a MaxCompute SQL statement cannot start with underscores (`_`). Therefore, grave accents (`'`) must be used. We recommend that you use aliases to avoid this issue.

```
select `'_c3`, `'_c4` from sale_detail_ctas3;
```

To ensure that the destination table has the same schema as the source table, use the following

`create table...like` statement:

```
create table sale_detail_like like sale_detail;
```

 **Note** The schema of `sale_detail_like` is exactly the same as that of `sale_detail`. Both 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`.

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

```
desc <table_name>;  
desc extended <table_name>;-- View table information and extended information.
```

MaxCompute allows you to use the `SHOW CREATE TABLE` statement to generate a DDL statement for table creation. This facilitates the SQL-based rebuild of the table schema.

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

6.3.1.2. Delete a table

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

Command syntax:

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

 **Note** If the command is run without the `IF EXISTS` option and the table does not exist, an exception is returned. With this option, a success is returned regardless of whether the table exists.

Example:

```
create table sale_detail_drop like sale_detail; drop table sale_detail_drop;  
-- If the table exists, a success is returned. If not, an exception is returned.  
drop table if exists sale_detail_drop2;  
-- A success is returned regardless of whether sale_detail_drop2 exists.
```

6.3.1.3. Rename a table

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

Command syntax:

```
alter table table_name rename to new_table_name;
```

-  **Note**
- The rename operation only changes the table name, not the table data.
 - If the table specified by `new_table_name` already exists, an error is returned.
 - If the table specified by `table_name` does not exist, an error is returned.

Example:

```
create table sale_detail_rename1 like sale_detail;  
alter table sale_detail_rename1 rename to sale_detail_rename2;
```

6.3.1.4. Modify the comment of a table

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

Command syntax:

```
alter table table_name set comment 'tbl comment';
```

Note

- table_name must be an existing table.
- A comment can contain a maximum of 1,024 bytes.

Example:

```
alter table sale_detail set comment 'new coments for table sale_detail';
```

You can run the desc command to view the modified comment in the table. For more information, see [Obtain table information](#).

6.3.1.5. Modify the lifecycle of a table

MaxCompute provides the lifecycle management function to release storage space and simplify the data clearance process. This topic describes how to run a DDL statement to modify the lifecycle of a table.

Command syntax:

```
alter table table_name set lifecycle days;
```

 Note

- The days parameter indicates the lifecycle of a table. Unit: days. It must be a positive integer.
- If the table specified by table_name is a non-partitioned table, and is not modified in the period specified by the days parameter since the last modification date, MaxCompute automatically clears the table (similar to the DROP TABLE operation). In MaxCompute, the LastDataModifiedTime value of a table is updated each time data in the table is modified. MaxCompute determines whether to clear a table based on its LastDataModifiedTime and lifecycle settings.
- If the table specified by table_name is a partitioned table, MaxCompute determines whether to clear each partition based on the LastDataModifiedTime value. Unlike non-partitioned tables, a partitioned table is not deleted after the last partition is reclaimed.
- You can configure a lifecycle for tables, but not for partitions.
- You can specify a lifecycle when creating a table.

Example:

```
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.
```

6.3.1.6. Disable or restore 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 execute DDL statements to disable or restore the lifecycle feature.

Syntax

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

 Note

- **TABLE DISABLE LIFECYCLE**
 - It prevents the reclamation of a table and its partitions based on the lifecycle feature. This option has a higher priority than partition_spec enable lifecycle.
 - The lifecycle settings and the partition_spec enable/disable flag of a table are retained.
 - You can still modify the lifecycle settings of a table and its partitions.
- **TABLE ENABLE LIFECYCLE**
 - After the lifecycle feature is enabled again, a table and its partitions can be reclaimed based on the lifecycle feature. By default, the lifecycle settings of the current table and its partitions are used.
 - Before you restore the lifecycle feature for a table and its partitions, you can configure new lifecycles for the table and its partitions. This prevents data from being mistakenly reclaimed due to the use of the previous settings.

Example

```
ALTER TABLE trans PARTITION(dt='20191111') DISABLE LIFECYCLE;
```

6.3.1.7. Modify the LastDataModifiedTime value of a table

MaxCompute SQL supports the TOUCH operation, which allows you to modify the LastDataModifiedTime value of a table. This operation changes the LastDataModifiedTime value of a table to the current time. This topic describes how to run a DDL statement to modify the LastDataModifiedTime value of a table.

Command syntax:

```
alter table table_name touch;
```

 Note

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

For more information about how to modify the LastDataModifiedTime value of a partition, see [Modify the LastDataModifiedTime value of a partition](#).

6.3.1.8. Clear data from a non-partitioned table

This topic describes how to run a DDL statement to clear data from a non-partitioned table.

Command syntax:

```
TRUNCATE TABLE table_name;
```

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

6.3.1.9. Archive table data

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

If a project does not have enough space, you can use the table archiving feature in MaxCompute to compress data by about 50%. The archiving feature uses a compression algorithm with a higher 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 effective storage ratio from 1:3 to 1:1.5. The archive feature consumes only half of the usual physical space.

However, this feature comes at a price. If a data block or machine is damaged, the time required to restore the data is longer, and the read performance is affected. Therefore, this feature is suitable for compressing cold data for storage. For example, you can store large volumes outdated log data as RAID files for a long time.

Command syntax:

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

Example:

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

Command 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 logical size and physical size during the archiving process. In the archiving process, multiple small files are automatically merged. After the archive operation is complete, you can run the `desc extended` command to check whether the data in the partition has been archived, and view the physical space usage:

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

6.3.1.10. Forcibly delete data from a table (partition)

If you need to forcibly and irreversibly delete data from a table or partition to immediately release storage space, you can perform the deletion operation with the PURGE option. This topic describes how to run a DDL statement to forcibly delete data from a table (partition).

Command syntax:

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

Example:

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

6.3.2. View-based operation

6.3.2.1. Create a view

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

Command 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. Views in MaxCompute are not materialized views. View operations involve accessing data of referenced tables. Note that changes to your permission on the referenced table can result in changes to your permission on the view.
- A view can contain only one valid SELECT statement.
- A view can reference other views but cannot reference itself. Circular reference is not supported.
- You cannot write data to a view. For example, the INSERT INTO and INSERT OVERWRITE operations 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 properly.
- If the CREATE VIEW statement is run without the IF NOT EXISTS option and the view already exists, an exception is returned. In this case, you can run the CREATE VIEW or REPLACE VIEW statement to recreate a view. The permissions on the recreated view remain unchanged.

Example:

```
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;
```

6.3.2.2. Delete a view

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

Command syntax:

```
drop view [if exists] view_name;
```

 **Note** If the command is run without the IF EXISTS option and the view does not exist, an error is returned.

Example:

```
drop view if exists sale_detail_view;
```

6.3.2.3. Rename a view

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

Command syntax:

```
alter view view_name rename to new_view_name;
```

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

Example:

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

6.3.3. Column and partition operations

6.3.3.1. Add a partition (ADD PARTITION)

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

Syntax

```
alter table table_name add [if not exists] partition partition_spec;-- Add a partition.
alter table table_name add [if not exists] partition partition_spec [PARTITION partition_spec PARTITION partition_spec...];-- Add multiple partitions at a time.
partition_spec:(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)
```

Note

- If you do not specify the IF NOT EXISTS option 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 partitioning column values.

Examples

The following examples show how to add partitions to the sale_detail table:

```
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 artition(region='shanghai');
-- Specify only the region partition. An error is returned.
```

6.3.3.2. Delete a partition (DROP PARTITION)

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

Syntax

```
alter table table_name drop [if exists] PARTITION partition_spec; -- Delete a partition.
alter table table_name drop [if exists] PARTITION partition_spec, PARTITION partition_spec, [
PARTITION partition_spec....] ;-- Delete multiple partitions at a time.
partition_spec:: (partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)
```

 **Note** If you do not specify the IF EXISTS option and the partition you want to delete does not exist, an error is returned.

Example

Execute the following statement to delete a partition from the sale_detail table:

```
alter table sale_detail drop partition(sale_date='201712',region='hangzhou');
-- The sales records of the China (Hangzhou) region for December 2017 are deleted.
```

6.3.3.3. Add a column

This topic describes how to add a column by using a DDL statement.

Command syntax:

```
alter table table_name add columns (col_name1 type1, col_name2 type2...)
```

 Note

- A column can only be one of the following types: bigint, double, boolean, datetime, decimal, string, tinyint, smallint, int, float, varchar, binary, timestamp, array, map, or struct.
- You can create up to 1,200 columns in a single table in MaxCompute.

6.3.3.4. Change a column name

This topic describes how to run a DDL statement to change a column name.

Command syntax:

```
alter table table_name change column old_col_name rename to new_col_name;
```

 Note

- You must specify an existing column for old_col_name.
- You cannot name a column in the table new_col_name.

6.3.3.5. Modify the comment of a column or partition

This topic describes how to run a DDL statement to modify the comment of a column or partition.

Command syntax:

```
alter table table_name change column col_name comment 'comment';
```

 Note

- The comment cannot exceed 1,024 bytes.
- The data type and position of a column cannot be changed.

6.3.3.6. Modify the LastDataModifiedTime value of a partition

MaxCompute SQL supports the TOUCH operation, which allows you to modify the LastDataModifiedTime value of a partition. This operation changes the LastDataModifiedTime value of a partition to the current time. This topic describes how to run a DDL statement to modify the LastDataModifiedTime value of a partition.

Command syntax:

```
alter table table_name touch partition(partition_col='partition_col_value', ...);
```

 Note

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

For more information about how to modify the LastDataModifiedTime value of a table, see [Modify the LastDataModifiedTime value of a table](#).

6.3.3.7. Modify partition values

MaxCompute SQL provides the RENAME operation, which allows you to modify partition values of a table. This topic describes how to run a DDL statement to modify partition values.

Command 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 command cannot modify the names of partition columns. It can only modify the values of the columns.
- To modify the values in one or more partitions in the case of multi-level partitions, you must specify values of partitions at each level.

6.3.3.8. Merge partitions

MaxCompute allows you to merge multiple partitions in a table into one partition and delete original partitions.

Syntax

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

Note

- If you do not specify the IF EXISTS option and the partition you want to merge does not exist, an error is returned.
- If you specify the IF EXISTS option but no partitions meet the merge conditions, no new partitions are generated.
- If source data is concurrently modified by operations such as INSERT, RENAME, or DROP when you execute the preceding statement, an error is returned even though you have specified the IF EXISTS option.
- If the PURGE attribute is specified, merged partitions cannot be restored by using the Kunlunjing.

Limits and troubleshooting

- External tables, shard tables, and tables with extreme storage are not supported. Xlib or Algo tables that depend on the file order are not supported. If you merge partitions of a clustered table, the clustered attribute is removed from the partitions.
- Hash operations are performed by CatalogServer on tables to merge partitions. A capacity limit is imposed on merged partitions. A hard link in the Apsara Distributed File System can have a maximum of seven replicas.
- You can merge a maximum of 4,000 partitions at a time.
- The number of partitions that can wait on CatalogServer to be merged is 10 million.
- If an error that indicates CatalogServer is busy occurs, try again later.
- If a hard link in the Apsara Distributed File System is faulty, purge the recycle bin and then try again.

Example

The following code shows the partitions and data of the tb_test table:

```
odps@ jet_zwz>list partitions tb_test;
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>read intpstringstringstring;
+-----+-----+-----+-----+
| value | ds      | hh      | mm      |
+-----+-----+-----+-----+
| 1      | 20181101 | 00      | 00      |
| 1      | 20181101 | 00      | 10      |
| 1      | 20181101 | 10      | 00      |
| 1      | 20181101 | 10      | 10      |
+-----+-----+-----+-----+
```

Execute the following statement to merge all partitions that meet the hh='00' condition into the ds=20181101/hh=00/mm=00 partition:

```
odps@ jet_zwz>alter table intpstringstringstring merge partition(hh='00') overwrite partiti
on(ds='20181101', hh='00', mm='00');
ID = 20190404025755844g80qwa7a
OK
```

Execute the following statement to view the partitions of the table after they are merged:

```
odps@ jet_zwz>list partitions intpstringstringstring;
ds=20181101/hh=00/mm=00
ds=20181101/hh=10/mm=00
ds=20181101/hh=10/mm=10
OK
```

Data in two partitions that meet the hh='00' condition is merged into the ds=20181101/hh=00/mm=00 partition.

```
odps@ jet_zwz>read intpstringstringstring;
+-----+-----+-----+-----+
| value | ds      | hh      | mm      |
+-----+-----+-----+-----+
| 1      | 20181101 | 00      | 00      |
| 1      | 20181101 | 00      | 00      |
| 1      | 20181101 | 10      | 00      |
| 1      | 20181101 | 10      | 10      |
+-----+-----+-----+-----+
```

When you merge partitions, you can specify multiple predicate conditions. For example, you can execute the following statement to merge all the partitions that remain to the ds=20181101/hh=00/mm=00 partition:

```
odps@ jet_zwz>alter table intpstringstringstring merge if exists partition(ds='20181101', h
h='00', mm='00'), partition(ds='20181101', hh='10', mm='00'), partition(ds='20181101', hh=
'10', mm='10') overwrite partition(ds='20181101', hh='00', mm='00') purge;
ID = 20190404034632854g431sqzt2
OK
odps@ jet_zwz>show partitions intpstringstringstring;
ds=20181101/hh=00/mm=00
OK
```

6.4. DML statements

6.4.1. INSERT statement

6.4.1.1. Update the data of a table

This topic describes how to run an INSERT statement to update the data of a table.

The INSERT OVERWRITE and INSERT INTO statements are commonly used for data processing in MaxCompute SQL. They are used to save the computing results in the target table for the next computing. The INSERT INTO statement adds data to a table or partition. The INSERT OVERWRITE statement clears the original data before inserting data to a table or partition.

Command syntax:

```
insert overwrite|into table tablename [partition (partcol1=val1, partcol2=val2 ...)] select
_statement
from from_statement;
```

 **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, not directly by the table name.

Example:

The following example calculates the sales of different regions in the sale_detail table.

```
create table sale_detail_insert like sale_detail;
alter table sale_detail_insert add partition(sale_date='2017', region='china');
insert overwrite table sale_detail_insert partition (sale_date='2017', region='china') sele
ct shop_name, customer_id, total_price from sale_detail;
```

 **Note** When data is updated using an INSERT operation, the mapping between the source and target tables depends on the column sequence in the SELECT clause, instead of the mapping of column names between both tables.

The following statement is also valid:


```
insert overwrite table sale_detail_insert partition (sale_date='2017', region='china')
select customer_id, shop_name, total_price from sale_detail;
-- When the sale_detail_insert table is created, the column sequence is shop_name string, c
ustomer_id string, and total_price bigint.
-- When data in sale_detail is inserted to sale_detail_insert, the insertion sequence is cu
stomer_id, shop_name, and total_price.
-- In this case, data in sale_detail.customer_id is inserted into sale_detail_insert.shop_n
ame.
-- Data in sale_detail.shop_name is inserted into sale_detail_insert.customer_id.
```

When data is inserted into a partitioned table, the partition columns cannot appear in the SELECT list.

```
insert overwrite table sale_detail_insert partition (sale_date='2017', region='china') sele
ct shop_name, customer_id, total_price, sale_date, region from sale_detail;
-- An error is returned, because partition columns (sale_date and region) cannot appear in
an INSERT statement for a static partition.
```

6.4.1.2. Output data to multiple objects

This topic describes how to run the INSERT statement to output data to multiple objects.

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

Command syntax:

```
from from_statement
insert overwrite | into table tablename1 [partition (partcol1=val1, partcol2=val2 ...)] sel
ect_statement1
[insert overwrite | into table tablename2 [partition ...] select_statement2]
```

Note

- A SQL statement typically supports up to 256 outputs. A syntax error is returned if more than 256 outputs are specified.
- In a MULTIINSERT statement, you can specify a target partition in a partitioned table or specify a non-partitioned table only once.
- The INSERT OVERWRITE and INSERT INTO operations cannot be performed simultaneously on different partitions in a partitioned table. Otherwise, an error is returned.

Example:

```
create table sale_detail_multi like sale_detail;
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2016', region='china' ) sele
ct shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2017', region='china' ) sele
ct shop_name, customer_id, total_price;
-- A success is returned. Data of the sale_detail table is inserted into the sale records o
f the China region in 2016 and 2017 in the sales table.
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2017', region='china' ) sele
ct shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2017', region='china' ) sele
ct shop_name, customer_id, total_price;
-- An error is returned. The same partition appears more than once.
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2016', region='china' )
select shop_name, customer_id, total_price
insert into table sale_detail_multi partition (sale_date='2017', region='china' ) select sh
op_name, customer_id, total_price;
-- An error is returned. The INSERT OVERWRITE and INSERT INTO operations cannot be performe
d simultaneously on different partitions in a partitioned table.
```

6.4.1.3. Output data to a dynamic partition

This topic describes how to use the INSERT statement to output data to a dynamic partition.

When you run the INSERT OVERWRITE statement on a partitioned table, you can specify the partition values in the statement. Another flexible method is to specify partition column names instead of setting partition values. In the meantime, specify the partition values in the corresponding columns of a SELECT clause.

Command syntax:

```
insert overwrite table tablename partition (partcol1, partcol2 ...) select_statement from f
rom_statement;
```

Note

- When you run a SQL dynamic partition statement in a distributed environment, a single process can output up to 512 dynamic partitions. If the number of dynamic partitions exceeds this limit, an exception is returned.
- Currently, a SQL dynamic partition statement can generate up to 2,000 dynamic partitions. If the number of dynamic partitions exceeds this limit, an exception is returned.
- The dynamic partition values cannot be NULL. Otherwise, an exception is returned.
- If a target 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.

Example:

```
create table total_revenues (revenue bigint) partitioned by (region string); insert overwrite table total_revenues partition(region)
select total_price as revenue, region from sale_detail;
```

 **Note** In the preceding example, you do not know which partitions are generated before running the SQL statement. The partitions generated are determined by the value of the region field after the execution of the SELECT statement. This is why the partitions are called dynamic partitions.

Other examples:

```
create table sale_detail_dypart like sale_detail;
insert overwrite table sale_detail_dypart partition (sale_date, region) select * from sale_detail;
-- A success is returned.
insert overwrite table sale_detail_dypart partition (sale_date='2017', region) select shop_name, customer_id, total_price, region from sale_detail;
-- A success is returned. The table has multi-level partitions. Specify a primary partition .
insert overwrite table sale_detail_dypart partition (sale_date='2017', region) select shop_name, customer_id, total_price from sale_detail;
-- An error is returned. The inserted dynamic partition must be in the SELECT list.
insert overwrite table sales partition (region='china', sale_date) select shop_name, customer_id, total_price, region from sale_detail;
-- An error is returned. You cannot specify only low-level partitions when dynamically inserting high-level partitions.
```

6.4.2. SELECT statement

6.4.2.1. SELECT

This topic describes how to use the SELECT statement.

Syntax

```
select [all | distinct] select_expr, select_expr, ... from table_reference
[where where_condition] [group by col_list]
[order by order_condition]
[distriute by distribute_condition [sort by sort_condition] ] [limit number]
```

Take note of the following points when you execute the SELECT statement:

- The SELECT statement reads data from a table. You can specify the names of the columns you want to read or use an asterisk (*) to represent all columns.

Examples

```
select * from sale_detail;
-- Read data from all columns in the sale_detail table.
select shop_name from sale_detail;
-- Read data from the shop_name column in the sale_detail table.
```

 **Note** The SELECT statement can only return a maximum of 1,000 rows of results. However, no such limits are imposed when SELECT is used as a clause. If SELECT is used as a clause, the clause returns all results in response to the query from the upper layer. To obtain more than 1,000 rows of results by using the SELECT statement, you must use Tunnel to download the entire table or a temporary table returned by the SELECT operation. For more information, see [MaxCompute Tunnel](#).

- You can use the WHERE clause to specify filter conditions.

Examples

```
select * from sale_detail where shop_name like 'hang%';
```

The following table describes filter conditions supported by the WHERE clause.

Filter conditions

Filter condition	Description
>, <, =, >=, <=, <>	/
like, rlike	/
in, not in	If a subquery is added after the IN or NOT IN condition, only one column is returned for the subquery and the number of return values cannot exceed 1,000.

You can specify partitions in the WHERE clause of the SELECT statement to avoid a full table scan.

Examples

```
select sale_detail.* from sale_detail
where sale_detail.sale_date >= '2015' and sale_detail.sale_date <= '2017';
```

 **Notice** To check whether partition pruning takes effect, execute the `EXPLAIN SELECT` statement. A common user-defined function (UDF) or the method that is used to specify partition conditions in a JOIN operation can cause partition pruning to fail to take effect.

UDFs support partition pruning. These UDFs are executed as small jobs and then replaced with the execution results.

You can use one of the following methods:

- 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 is used with limits. This method backfills partitions with execution results. A maximum of 1,000 partitions can be backfilled. If an annotation is added to the UDF class, an error that indicates more than 1,000 partitions are backfilled may be returned. If you want to ignore the error, add the `set odps.sql.udf.ppr.to.subquery = false;` flag to disable this feature globally. After this feature is disabled, UDF-based partition pruning becomes invalid.

The WHERE clause in an SQL statement can include the BETWEEN...AND condition. Example:

```
SELECT sale_detail.* FROM sale_detail
WHERE sale_detail.sale_date between '2017' and '2019';
```

Note The number of conditions that can be specified in the WHERE clause cannot exceed 256.

- Nested subqueries are supported in table_reference.

Examples

```
select * from (select region from sale_detail) t where region = 'shanghai';
```

- **DISTINCT:** If duplicate rows exist, add DISTINCT before the field to remove duplicate values. In this case, only one value is returned. If you use ALL, all duplicate values are returned. If you do not specify the DISTINCT option, the statement returns all duplicate values, same as the result obtained by using the ALL option.

Examples

```
select distinct region from sale_detail;
select distinct region, sale_date from sale_detail;
-- The DISTINCT option applies to multiple columns. The option takes effect on all columns of the SELECT statement, instead of a single column.
```

- **GROUP BY:** This clause is used to perform group-based queries. In most cases, this clause is used with aggregate functions. If a SELECT statement includes aggregate functions, the key of the GROUP BY clause can be the names of columns in the input table or an expression composed of input table columns. The key cannot be the aliases of the columns in the output table of the SELECT operation.

Note If the `set hive.groupby.position.alias=true;` flag is added before SQL statements, integer constants in the GROUP BY clause are considered column numbers in a SELECT operation. Example:

```
-- The columns in the sale_detail table are in the format of key-value pairs.
select region, sum(total_price) from sale_detail group by 1;
-- Equivalent to the following statement:
select region, sum(total_price) from sale_detail group by region;
```

Examples

```

select region from sale_detail group by region;
-- The statement is successfully executed because the name of a column in the input table
is used as the column in the GROUP BY clause.
select sum(total_price) from sale_detail group by region;
-- The statement is successfully executed because the values in the region column are use
d to group the table and to return the total sales of each group.
select region, sum(total_price) from sale_detail group by region;
-- The statement is successfully executed because the values in the region column are use
d to group the table and to return the unique region value and the total sales of each gr
oup.
select region as r from sale_detail group by r;
-- An error is returned because the alias of the column in the SELECT operation is used a
s the column in the GROUP BY clause.
select 'China-' + region as r from sale_detail group by 'China-' + region;
-- A complete expression of the column is required.
select region, total_price from sale_detail group by region;
-- An error is returned because all the columns that do not include aggregate functions i
n the SELECT operation must exist in the GROUP BY clause.
select region, total_price from sale_detail group by region, total_price;
-- The statement can be successfully executed.

```

 **Note** The GROUP BY operation is performed before the SELECT operation during the parsing of SQL statements. Therefore, GROUP BY uses only the column names or expressions of the input table as keys. For more information about aggregate functions, see [Aggregate functions](#).

- **ORDER BY:** This clause is used for global sorting based on specific columns. To sort records in descending order, use the DESC keyword. The ORDER BY clause must be used with the LIMIT clause because records are globally sorted. In an ORDER BY operation, NULL is considered the lowest of all values. This rule is consistent with MySQL, but is different from Oracle. Different from the GROUP BY clause, 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.

 **Note** 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. Example:

```

-- The columns in the sale_detail table are in the format of key-value pairs.
select region, sum(total_price) from sale_detail order by 2 limit 100;
-- Equivalent to the following statement:
select region, sum(total_price) from sale_detail order by sum(total_price) limit 100;

```

Examples

```

select * from sale_detail order by region;
-- An error is returned because the ORDER BY clause is not used with the LIMIT clause.
select * from sale_detail order by region limit 100;
select region as r from sale_detail order by region;
-- An error is returned because the ORDER BY clause is not followed by a column alias.
select region as r from sale_detail order by r;

```

 **Note** The number in the LIMIT clause is a constant that limits the number of output rows. If a SELECT statement is executed without the LIMIT clause, it can return a maximum of 5,000 rows. The screen display limit may vary with projects and can be configured in the console.

The OFFSET clause can be used with the ORDER BY LIMIT clause to skip the number of rows specified by OFFSET.

Examples

```
SELECT * FROM src ORDER BY key LIMIT 20 OFFSET 10;
-- Sort the rows of the src table in ascending order by key, and return the 11th to 30th rows. OFFSET 10 indicates that the first 10 rows are skipped, and LIMIT 20 indicates that a maximum of 20 rows can be returned.
```

- **DISTRIBUTE BY:** This clause is used to shard data based on hash values of specific columns. The DISTRIBUTE BY clause must be followed by the alias of an output column from the SELECT operation.

Examples

```
select region from sale_detail distribute by region;
-- The statement is successfully executed because the column name is used as the column alias.
select region as r from sale_detail distribute by region;
-- An error is returned because the DISTRIBUTE BY clause is not followed by a column alias.
select region as r from sale_detail distribute by r;
```

- **SORT BY:** This clause is used for partial sorting. The DISTRIBUTE BY clause must be placed before the SORT BY clause. In practice, the SORT BY clause is used to partially sort the results of the DISTRIBUTE BY clause. The SORT BY clause must be followed by the alias of an output column from the SELECT operation.

Examples

```
select region from sale_detail distribute by region sort by region; select region as r from sale_detail sort by region;
-- An error is returned because the SORT BY clause does not follow a DISTRIBUTE BY clause.
```

- The ORDER BY and GROUP BY clauses cannot be used with the DISTRIBUTE BY and SORT BY clauses. The ORDER BY and GROUP BY clauses must be followed by the alias of an output column from the SELECT operation.

 **Note**

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

6.4.2.2. Subquery

This topic describes how to use the SELECT statement for subquery operations.

A common SELECT statement reads data from multiple tables, for example, select column_1, column_2 ... from table_name. The query object can be another SELECT operation, which is a subquery.

Command syntax:

```
select * from (select shop_name from sale_detail) a;
```

 **Notice** A subquery must have an alias.

Example:

```
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.

6.4.3. UNION statements

6.4.3.1. UNION ALL

This topic describes how to execute the SELECT statement to perform the UNION ALL operation.

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 the combination of two top-level query results. To combine them, rewrite them into a subquery.

Format example before rewriting

```
select * from sale_detail where region = 'hangzhou'
union all
select * from sale_detail where region = 'shanghai';
```

Format example 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.

Example:

```
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 c
lause for the src table based on the obtained result.
```

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.

6.4.4. JOIN statement

6.4.4.1. JOIN

This topic describes how to use a JOIN statement.

MaxCompute supports multiple JOIN operations in an SQL statement. JOIN does not support Cartesian products (JOIN without an ON clause).

Syntax

```
join_table:
table_reference join table_factor [join_condition]
| table_reference {left outer|right outer|full outer|inner} join table_reference join_condi
tion
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.

Take note of the following points when you perform a JOIN operation:

- LEFT OUTER JOIN: returns all rows in the left table, such as shop in the following example. The returned rows include the rows that do not match any rows in the right table, such as sale_detail in the following example.

Example

```
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.
```

- **RIGHT OUTER JOIN:** returns all rows in the right table, such as sale_detail in the following example. The returned rows include the rows that do not match any rows in the left table, such as shop in the following example.

Example

```
select a.shop_name as ashop, b.shop_name as bshop from shop a right outer join sale_detail
l 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.
```

- **FULL OUTER JOIN:** returns all rows in both the left and right tables.

Example

```
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:** only returns the rows in which two tables can be mapped. The INNER keyword can be omitted.

Example

```
select a.shop_name from shop a inner join sale_detail b on a.shop_name=b.shop_name; select
t a.shop_name from shop a join sale_detail b on a.shop_name=b.shop_name;
```

- **Join condition:** You must use equi-joins and combine conditions by using AND. A maximum of 128 JOIN operations are supported in an SQL statement. You can use non-equi joins or combine conditions by using OR in a MAPJOIN operation.

Example

```
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;
-- A maximum of 128 JOIN operations are supported in an SQL statement.
select a.* from shop a join sale_detail b on a.shop_name <> b.shop_name;
-- An error is returned because MaxCompute does not support non-equi joins.
```

- **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.

Example

```
-- To join the src table that contains the key1, key2, a1, and a2 columns and the src2 table
that contains the key1, key2, b1, and b2 columns, execute the following statement:
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;
```

The syntax that uses a pair of parentheses to specify the priorities of JOIN operations is supported.

Example

```
SELECT * FROM src JOIN (src2 JOIN src3 on xxx) ON yyy;  
-- The src2 JOIN src3 operation is executed first. Then, the JOIN operation is performed on  
the src table based on the result.
```

6.4.4.2. MAPJOIN HINT

This topic describes how to use a MAPJOIN statement to join a large table with one or more small tables.

A MAPJOIN operation is faster than common JOIN operations.

When the volume of data is small, MAPJOIN accelerates the execution process by using SQL to load all the specified small tables into the program memory through the JOIN operation.

Example

```
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;
```



Notice

Note the following points when you use a MAPJOIN statement:

- The left table of a LEFT OUTER JOIN clause must be a large table.
- The right table of a RIGHT OUTER JOIN clause must be a large table.
- Both the left and right tables of an INNER JOIN clause can be large tables.
- MAPJOIN cannot be used in a FULL OUTER JOIN clause.
- MAPJOIN supports small tables in subqueries.
- If you need to reference a small table or a subquery when using MAPJOIN, you must reference the alias of the table or subquery.
- In MAPJOIN, you can use non-equi joins or combine multiple conditions by using OR.
- If MAPJOIN is used, the total memory occupied by all the small tables cannot exceed 512 MB. However, you can use the *odps.sql.mapjoin.memory.max* parameter to raise this limit up to 2,048 MB.

The limit here refers to the original size of data. If you run the desc command to obtain the compressed size, you must multiply it by the compression ratio.

In MaxCompute SQL, you cannot use non-equi joins or the OR logic in the ON condition. However, you can do this in MAPJOIN. Example:

```
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;
```

6.4.5. EXPLAIN statement

This topic describes the EXPLAIN statement in DML statements of MaxCompute SQL.

MaxCompute SQL provides the EXPLAIN operation, which displays the description of the ultimate execution plan structure of DML statements. An execution plan is the program that is ultimately used to execute SQL semantics.

Command syntax:

```
EXPLAIN <DMLquery>;
```

Note

The execution result of an EXPLAIN statement includes the following:

- Dependencies between all the jobs of this DML statement.
- Dependencies between all the tasks of each job.
- All operator dependency structures in a task.

Example:

```
EXPLAIN
SELECT abs(a.key), b.value FROM src a JOIN src1 b ON a.value = b.value;
```

The EXPLAIN statement output includes the following:

- The first part is the dependency between jobs.

Command output:

```
job0 is root job
```

 **Note** Because this query only needs one job (job0), only one line of information is needed.

- The second part is the dependency between tasks.

Command output:

```
In Job job0:
root Tasks: M1_Stg1, M2_Stg1
J3_1_2_Stg1 depends on: M1_Stg1, M2_Stg1
```

Note

- Job0 contains three tasks, among which M1_Stg1 and M2_Stg1 are executed first, and J3_1_2_Stg1 is executed after the first two tasks are finished.

- Naming rules for tasks: MaxCompute provides four task types: MapTask, ReduceTask, JoinTask, and LocalWork. The first letter of a task name indicates the type of the current task (for example, M2Stg1 is a MapTask). The number immediately following the first letter represents the current Task ID, which is unique among all tasks in the current query. The numbers separated by underscores (_) represent the immediate dependencies of the current task. For example, J3_1_2_Stg1 means that the current task (ID 3) is dependent on tasks with ID 1 and ID 2.
- The third part is the operator structure in the tasks, where each operator string describes the execution semantics of a task.

Command output:

```
In Task M1_Stg1:
Data source: yudi_2.src #### "Data source" describes the input content of the current task
TS: alias: a #### TableScanOperator
RS: order: + #### ReduceSinkOperator keys:
a.value values:
a.key partitions:
a.value
In Task J3_1_2_Stg1:
JOIN: a INNER JOIN b #### JoinOperator
SEL: Abs(UDFToDouble(a._col0)), b._col5 #### SelectOperator FS: output: None #### FileSinkOperator
In Task M2_Stg1:
Data source: yudi_2.src1 TS: alias: b
RS: order: + keys:
b.value values:
b.value partitions:
b.value
```

The meanings of the operators are shown as below.

Operators

Operator	Description
TableScanOperator	Describes the logic of FROM statement blocks in a query statement. The input table name (alias) is displayed in the EXPLAIN results.
SelectOperator	Describes the logic of SELECT statement blocks in a query statement. The columns passed to the next operator, separated by commas, are displayed in the EXPLAIN results. If the result is a reference to a column, it is displayed as < alias >. < column_name >. If the result is an expression, it is displayed as a function, for example, func1(arg1_1, arg1_2, func2(arg2_1, arg2_2)). If the result is a constant, the value is displayed directly.
FilterOperator	Describes the logic of WHERE statement blocks in a query statement. A WHERE condition, which complies with a display rule similar to that of selectOperator, is displayed in the EXPLAIN results.

Operator	Description
JoinOperator	Describes the logic of JOIN statement blocks in a query statement. The tables involved in the JOIN operation and the mode of JOIN operation are displayed in the EXPLAIN results.
GroupByOperator	Describes the logic of the AGGREGATE operation. This structure 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 the data distribution operation between tasks. If the result of the current task is transferred to another task, ReduceSinkOperator must be used to distribute data at the end of the current task. The output sorting method, the distributed keys, values, and columns used to calculate the hash value are displayed in the EXPLAIN results.
FileSinkOperator	Describes the final data storage operation. If there is an INSERT statement block in the query statement, the name of the target table is displayed in the EXPLAIN results.
LimitOperator	Describes the logic of LIMIT statement blocks in a query statement. The limit value is displayed in the EXPLAIN results.
MapjoinOperator	Describes JOIN operations in large tables, similar to JoinOperator.

 Note

- If a query is complex and has too many EXPLAIN results, the API restriction is triggered, and incomplete results are displayed. In this case, the query can be split, and the EXPLAIN operation can be performed on each part to show the structure of the job.
- The maximum number of partitions in a query is 10,000. Inputting too many partitions leads to over-length Data source content. To circumvent this limit, you can filter out most partitions by adding a query filter.

6.4.6. GROUPING SETS

6.4.6.1. Overview

For scenarios where you need to aggregate and analyze data of multiple dimensions, you must execute multiple UNION ALL clauses. For example, you wanted to aggregate column a, aggregate column b, and aggregate columns a and b together. The GROUPING SETS clause is a better choice in such cases.

GROUPING SETS is an extension to the GROUP BY clause in the SELECT statement. You can group results in various ways by using GROUPING SETS without executing multiple SELECT statements. This can produce better execution plans and result in higher performance from the MaxCompute engine.

 **Notice** Many examples in this topic are demonstrated using MaxCompute Studio. We recommend that you install MaxCompute Studio before you proceed with subsequent operations.

6.4.6.2. Example

The following example is for your reference.

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. Use GROUPING SETS.

```
SELECT os,device, city ,COUNT(*)
FROM requests
GROUP BY os, device, city GROUPING SETS((os, device), (city), ());
```

A similar output is displayed.

Command output

	os	device	city	_c3
1	ios	Phone	\N	1
2	linux	PC	\N	1
3	linux	Phone	\N	1
4	windows	PC	\N	3
5	windows	Phone	\N	1
6	\N	\N	Beijing	4
7	\N	\N	Shijiazhuang	3
8	\N	\N	\N	7

 **Note** 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 GROUPING SETS method is simpler and more efficient.

 **Notice** Expressions not used in GROUPING SETS use NULL as placeholders. You can execute UNION statements on grouping sets.

6.4.6.3. CUBE and ROLLUP

CUBE and ROLLUP are special GROUPING SETS functions. CUBE lists all possible combinations of the specific columns as grouping sets. ROLLUP aggregates data by level to generate grouping sets.

Example

```
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), ()
)
```

6.4.6.4. GROUPING and GROUPING_ID

NULL is used as placeholders in grouping sets, but it can also be a value that is manually entered. In the code, however, placeholder NULLs are indistinguishable from value NULLs. The GROUPING function is provided to address this issue.

GROUPING allows you to specify the name of a column as a parameter. If the specified lines are aggregated based on a column whose name is used as a parameter in this function, 0 is returned, indicating that NULL is an entered value. Otherwise, 1 is returned, indicating that NULL is a placeholder.

GROUPING_ID can be used to specify the names of one or more columns as parameters. The GROUPING results in these columns are formed into integers by using BitMap.

Example:

```
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);
```

A similar output is displayed.

Command output

	a	b	c	_c3	ga	gb	gc	groupingid
1	W	W	W	1	1	1	1	7
2	W	W	3	1	1	1	0	6
3	W	2	W	1	1	0	1	5
4	W	2	3	1	1	0	0	4
5	1	W	W	1	0	1	1	3
6	1	W	3	1	0	1	0	2
7	1	2	W	1	0	0	1	1
8	1	2	3	1	0	0	0	0

6.4.7. IF statement

MaxCompute SQL supports the IF-ELSE statement.

You can use the IF-ELSE statement to execute SQL scripts with specific conditions. The 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 because 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. SELECT TRANSFORM

6.5.1. Overview

SELECT TRANSFORM implements features that MaxCompute SQL does not provide. SELECT TRANSFORM allows you to start a specified child process and enter data of a required format into the child process through standard input (stdin). Then, you can parse the standard output (stdout) of the child process to obtain the final output. This process does not require you to compile UDFs.

SELECT TRANSFORM simplifies the reference of script code and supports programming languages such as Java, Python, Shell, and Perl. It is suitable for ad hoc data analysis. MaxCompute Select Transform is fully compatible with Hive syntax, features, and actions, including input/output row format and reader/writer. Most Hive scripts can be added directly to the SELECT TRANSFORM statement. Others can be used after a few changes.

Command 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)?)?
```

Description:

- **SELECT TRANSFORM:** The **SELECT TRANSFORM** keyword can be replaced with the **MAP** or **REDUCE**

keyword while maintaining the same semantic meaning. However, we recommend that you use **SELECT TRANSFORM** because its syntax is simpler.

- (arg1, arg2 ...): arguments in the TRANSFORM clause. Their format is similar to those of items in the SELECT clause. In the default format, the results of expressions for each argument are combined by using `\t` after they are implicitly converted into strings. The arguments are then entered into the specified child process.

 **Note** The default format is configurable. For more information, see **ROW FORMAT**.

- **USING**: specifies the command used to start a child process. Note the following points about the USING clause.
 - In most MaxCompute SQL statements, the USING clause can only specify resources. However, in the SELECT TRANSFORM statement, the USING clause can specify commands to ensure compatibility with Hive syntax.
 - The format of the USING clause is similar to the syntax of a Shell script. However, a Shell script is not actually expected to start the child process. The child process is created based on the command input. Because of this, a number of Shell functions, such as input and output redirection, pipe, and loop, are unavailable. A Shell script can be used as to start a child process if necessary.
- **RESOURCES**: specifies 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;` clause before SQL statements.

 **Notice** This clause takes effect globally once it is specified. All SELECT TRANSFORM statements will be able to access the resources specified by this clause.

- **ROW FORMAT**: specifies the input or output format. Two ROW FORMAT clauses are used in the syntax: the first one specifies the input format, and the second one specifies the output format. `\t` is used to separate columns, `\n` is used to separate rows, and NULL is represented by `\N`.

 Notice

- For `field_delimiter`, `character_escape`, and `line_separator`, only one character can be accepted. If you specify a string, the first character in the string takes priority over the others.
- There are a variety of Hive syntaxes to specify formats. MaxCompute supports syntaxes such as `inputRecordReader`, `outputRecordReader`, and `Serdeinput`. To use these formats, you must enable Hive compatibility by adding the `set odps.sql.hive.compatible=true;` clause before SQL statements. If you specify a syntax such as `inputRecordReader` or `outputRecordReader` supported by Hive, statements may be executed at lower speeds.

- **AS**: specifies output columns.

 Note

- You can specify data types in the AS clause, as in `as(col1:bigint, col2:boolean)`. By default, strings are returned if you do not specify data types, as in `as(col1, col2)`.
- The output is obtained by parsing the stdout of the child process. If the specified data types do not include STRING, the system implicitly calls the CAST function. Runtime exceptions may occur when the CAST function is called.
- You cannot specify data types for only some of the columns, as in `as(col1, col2:bigint)`.
- If you skip the AS clause, the field preceding the first `\t` in the stdout is a key, and all the following parts are a value. This is equivalent to `as(key, value)`.

6.5.2. SELECT TRANSFORM examples

6.5.2.1. Call Shell scripts

In this example, a Shell script is used to generate 50 lines of data starting from 1 to 50. The output of the data field is as follows:

```
SELECT  TRANSFORM(script) USING 'sh' AS (data)
FROM    (
          SELECT  'for i in `seq 1 50`; do echo $i; done' AS script
        ) t
;
```

The Shell commands are used as the input of the TRANSFORM clause.

 **Note** In addition to language extensions, SELECT TRANSFORM also provides simple features of AWK, Python, Perl, and Shell to compile scripts in commands. You do not need to compile script files or upload resources separately.

You can upload script files for complex cases, as in the following example Python script call.

6.5.2.2. Call Python scripts

This topic provides an example of how to use SELECT TRANSFORM to call Python scripts.

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 int(token[0]) + int(token[1])
    line = sys.stdin.readline()
```

2. Add the Python script file as a resource to MaxCompute.

```
add py ./myplus.py -f;
```

 **Note** You can also add resources from the DataWorks console.

3. Execute the SELECT TRANSFORM statement to call the resource.

```
Create table testdata(c1 bigint,c2 bigint); -- Create a test table.
insert into Table testdata values (1,4),(2,5),(3,6); -- Insert test data into the test
table.
-- Execute the SELECT TRANSFORM statement:
SELECT
TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py' resources 'myplus.py'
AS (result bigint)
FROM testdata;
-- Or
set odps.sql.session.resources=myplus.py;
SELECT
TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py'
AS (result bigint)
FROM testdata;
```

4. A similar output is displayed:

```
+-----+
| cnt |
+-----+
| 5   |
| 7   |
| 9   |
+-----+
```

Python scripts are not subject to any format requirements and do not require a Python framework to be run in MaxCompute. In MaxCompute, Python commands can be used as the input of the TRANSFORM clause. For example, you can call Shell scripts by running Python commands.

```
SELECT TRANSFORM('for i in xrange(1, 50): print i;') USING 'python' AS (data);
```

6.5.2.3. Call Java scripts

Java scripts are called in a similar manner to Python scripts. In this example, you need to compile a Java script file, export it as a JAR package, and then run the add command to add the JAR package as a resource to MaxCompute. The resource will be called by using SELECT TRANSFORM.

1. Compile a Java script file and export it as a JAR package. In this example, the name of the JAR package is Sum.jar.

```

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 table testdata(c1 bigint,c2 bigint); -- Create a test table.
insert into Table testdata values (1,4),(2,5),(3,6); -- Insert test data into the test
table.
-- 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;
-- Or
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;

```

4. A similar output is displayed:

```

+-----+
| cnt |
+-----+
| 5   |
| 7   |
| 9   |
+-----+

```

You can use the preceding method to run most Java utilities.

Although UDTF frameworks are provided for Java and Python, it is easier to compile code by using SELECT TRANSFORM. SELECT TRANSFORM is a simpler process because it is not subject to any 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.

6.5.2.4. Call scripts of other languages

In addition to language extensions, SELECT TRANSFORM also supports commonly used Unix command and script interpreters, such as AWK and Perl.

An example of calling AWK:

```
SELECT TRANSFORM(*) USING "awk '{print $2}'" as (data) from testdata;
```

An example of calling Perl:

```
SELECT TRANSFORM (testdata.c1, testdata.c2) USING "perl -e 'while($input = <STDIN>){print $input;}'" FROM testdata;
```



Notice PHP and Ruby are not deployed in the MaxCompute cluster and cannot be called.

6.5.2.5. Call scripts in series

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;
```

More often, you can use either the map or reduce keywords to produce 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;
```

6.5.3. Performance advantages

The performance of SELECT TRANSFORM and UDTF varies depending on the specific scenario. In general, SELECT TRANSFORM performs better. However, UDTF performs better as the volume of data increases. Because the development of transform is easier, SELECT TRANSFORM is more suitable for ad hoc data analysis.

The advantages of UDTFs and SELECT TRANSFORM are listed in the following sections.

Advantages of UDTFs

- Output and input follow specified data types and do not require conversion.
- Processes are not suspended if the operating system pipe is empty or fully occupied. The operating system pipe has a 4 KB buffer.
- Constant parameters do not need to be transmitted.

Advantages of SELECT TRANSFORM

- Supports child and parent processes and can utilize multiple server cores when high CPU usage and low throughput is needed.
- Calls underlying systems to read and write data to be transmitted, giving it a higher performance than Java.
- Supports tools such as AWK and can run native code.

6.6. UNION, INTERSECT, and EXCEPT

This topic describes SQL syntax, descriptions and examples of UNOIN ALL, UNION DISTINCT, INTERSECT ALL, INTERSECT DISTINCT, 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;
```

Purpose: It is used to return the union of two data sets, the intersection of two data sets, or the complement of the second dataset in the first dataset.

Description:

- UNION: returns the union of two datasets. It combines the two datasets into one dataset.
- INTERSECT: returns the intersection of two datasets. It outputs the records contained in both datasets.
- EXCEPT: returns the complement of the second dataset in the first dataset. It outputs the records that are contained in the first dataset, but not in the second dataset.
- MINUS: equivalent to EXCEPT.

Examples:

- UNOIN ALL example:

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)  
UNION ALL  
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);
```

Returned result: two datasets are combined.

a	b
1	2
1	4
1	2
1	2
3	4

- **UNION DISTINCT example:**

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)
UNION
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);
```

Returned result: equivalent to `SELECT DISTINCT * FROM (< the result of UNION ALL >) t; .`

a	b
1	2
1	4
3	4

- **INTERSECT ALL example:**

```
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);
```

Returned result: deduplication is skipped in INTERSECT ALL. It seems that there is a hidden serial number behind the same row and each row can be displayed separately.

a	b
1	2
1	2
3	4

- **INTERSECT DISTINCT example:**

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
INTERSECT
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);
```

Returned result: `SELECT DISTINCT * FROM (< the result of INTERSECT ALL >) t; .`

a	b
1	2
3	4

● EXCEPT ALL example:

```
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);
```

Returned result: deduplication is skipped in EXCEPT ALL. There is a hidden serial number behind the same row and each row can be displayed separately.

a	b
1	2
1	2
3	4
7	8

● EXCEPT DISTINCT example:

```
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);
```

Returned result: equivalent to `Select distinct * FROM left_branch limit t all select distinct * FROM right_branch;`

a	b
1	2
7	8

Note

- Sorting may be skipped in the preceding operations.
- The left and right branches in the preceding operations must have the same number of columns. In addition, if data types in the left and right branches are not consistent, they may be implicitly converted. Due to compatibility issues, implicit conversion is not carried out between STRING and no-STRING types for the preceding operations.
- Up to 256 branches are allowed in the preceding operations. An error is returned if more branches are used.
- If the UNION statement is followed by the CLUSTER BY, DISTRIBUTE BY, SORT BY, ORDER BY or LIMIT clause and you add `set odps.sql.type.system.odps2=false;`, the SET statement is applicable to the last `select_statement;` of the UNION statement. If you add `set odps.sql.type.system.odps2=true;`, the SET statement is applicable to all `select_statements` of the UNION statement. Example:

```
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;
```

Returned result:

```
+-----+
| a      |
+-----+
| 0      |
| 1      |
| 2      |
+-----+
```

6.7. Built-in functions

6.7.1. Mathematical functions

6.7.1.1. ABS

This topic describes the ABS function.

Function declaration:

```
double abs(double number)
bigint abs(bigint number)
decimal abs(decimal number)
```

Purpose: It is used to return absolute values.

Description:

number: double, bigint or decimal type. When the input is of the bigint type, a value of the bigint type is returned; when the input is of the double type, a value of the double type is returned. If the input is of the string type, it is implicitly converted into a value of the double type before this computation. If the input is of another type, an error is returned.

Returned value: double, bigint, or decimal type, depending on the type of the input. If the input is NULL, NULL is returned.

 **Note** When the input is of the bigint type and is out of the maximum range of the bigint type, the returned value is of the double type. In this case, the precision may be diminished.

Example:

```
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 a complete ABS function in SQL. Other built-in functions (except window functions and aggregation functions) are in similar usage to this function and are not shown here.

```
select abs(id) from tbl1;
-- Take the absolute value of the id field in tbl1.
```

6.7.1.2. ACOS

Function declaration:

```
double acos(double number)
decimal acos(decimal number)
```

Purpose: It is used to calculate the arccosine of a number.

Description:

number: double or decimal type. Value range: -1 to 1. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double or decimal type. Value range: 0 to π . If number is NULL, NULL is returned.

Example:

```
acos("0.87") = 0.5155940062460905
acos(0) = 1.5707963267948966
```

6.7.1.3. ASIN

Function declaration:

```
double asin(double number)
decimal asin(DECIMAL number)
```

Purpose: It is used to calculate the arcsine of a number.

Description:

number: double or decimal type. Value range: -1 to 1. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double or decimal type. Value range: $-\pi/2$ to $\pi/2$. If number is NULL, NULL is returned.

Example:

```
asin(1) = 1.5707963267948966
asin(-1) = -1.5707963267948966
```

6.7.1.4. ATAN

Function declaration:

```
double atan(double number)
```

Purpose: It is used to calculate the arctangent of a number.

Description:

number: double type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double type. Value range: $-\pi/2$ to $\pi/2$. If number is NULL, NULL is returned.

Example:

```
atan(1) = 0.7853981633974483;
atan(-1) = -0.7853981633974483
```

6.7.1.5. CEIL

Command syntax:

```
bigint ceil(double value)
bigint ceil(decimal value)
```

Purpose: It is used to return the smallest integer that is equal to or greater than the input value.

Description:

value: double or decimal. If the value is of the string or bigint type, it is implicitly converted to the double type. For all other input types, an error is returned.

Returned value: bigint type. If the input is NULL, NULL is returned.

Example:

```
ceil(1.1) = 2  
ceil(-1.1) = -1
```

6.7.1.6. CONV

Command syntax:

```
string conv(string input, bigint from_base, bigint to_base)
```

Purpose: It is used to convert a number from one numeric base number system to another.

Description:

- **input:** an integer of the string type to be converted. It accepts values of the bigint and double types by means of implicit conversion.
- **from_base, to_base:** a number system value in decimal form. Value range: 2, 8, 10, and 16. It accepts values of the string and double types by means of implicit conversion.

Returned value: string type. If any input is NULL, NULL is returned. The conversion process runs at a 64-bit precision. An error is returned when overflow occurs. If the input is a negative value (beginning with '-'), an error is returned. If the input is a decimal, it is converted to an integer before hex conversion. The decimal part is left out.

Example:

```
conv('1100', 2, 10) = '12'  
conv('1100', 2, 16) = 'c'  
conv('ab', 16, 10) = '171'  
conv('ab', 16, 16) = 'ab'
```

6.7.1.7. COS

Command syntax:

```
double cos(double number)  
decimal cos(decimal number)
```

Purpose: It is used to return the cosine of a number. The input must be a radian value.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted to a value of the double type. For all other input types, an error is returned.

Returned value: double or decimal type. If the input is NULL, NULL is returned.

Example:

```
cos(3.1415926/2) = 2.6794896585028633e-8  
cos(3.1415926) = -0.9999999999999986
```

6.7.1.8. COSH

Command syntax:

```
double cosh(double number)
decimal cosh(decimal number)
```

Purpose: It is used to return the hyperbolic cosine of a number.

Description:

number: double or decimal. If the input is of the string or bigint type, it is implicitly converted to a value of the double type. For all other input types, an error is returned.

Returned value: double or decimal. If the input is NULL, NULL is returned.

6.7.1.9. COT

Function declaration:

```
double cot(double number)
decimal cot(decimal number)
```

Purpose: It is used to return the cotangent of a number. The input must be a radian value.

Description:

number: double or decimal. If the input is of the string or bigint type, it is implicitly converted a value of the double type. For all other input types, an error is returned.

Returned value: double or decimal type. If the input is NULL, NULL is returned.

6.7.1.10. EXP

Function declaration:

```
double exp(double number)
decimal exp(decimal number)
```

Purpose: It is used to return the exponent value of number.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double or decimal type. If number is NULL, NULL is returned.

6.7.1.11. FLOOR

Function declaration:

```
bigint floor(double number)
bigint floor(decimal number)
```

Purpose: It is used to return the round-down integer that is less than or equal to number.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: bigint type. If number is NULL, NULL is returned.

Example:

```
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
```

6.7.1.12. LN

Function declaration:

```
double ln(double number)
decimal ln(decimal number)
```

Purpose: It is used to return the natural logarithm of a number.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double or decimal type. If the input is NULL, negative, or zero, NULL is returned.

6.7.1.13. LOG

Function declaration:

```
double log(double base, double x)
decimal log(decimal base, DECIMAL x)
```

Purpose: It is used to return the logarithm of x to base.

Description:

- base: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.
- x: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: logarithm value of the double or decimal type. If either base or x is NULL, negative, or zero, NULL is returned. If base is 1 (which leads to division by zero), NULL is returned.

6.7.1.14. POW

Command syntax:


```
double pow(double x, double y)
decimal pow(decimal x, decimal y)
```

Purpose: It is used to return the yth power of x, that is, x^y .

Description:

- x: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.
- y: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double or decimal type. If x or y is NULL, NULL is returned.

6.7.1.15. RAND

Command syntax:

```
double rand(bigint seed)
```

Purpose: It is used to return a random number of the double type from 0 to 1 based on the seed.

Description:

Seed: optional, bigint type. It is the seed of a random number, and determines the start value of the random number sequence.

Returned value: double type.

Example:

```
select rand() from dual;
select rand(1) from dual;
```

6.7.1.16. ROUND

Function declaration:

```
double round(double number, [bigint decimal_places])
decimal round(decimal number, [bigint decimal_places])
```

Purpose: It is used to return a number rounded to the specified decimal place.

Description:

- number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. If the input is of another type, an error is returned.
- decimal_place: a constant of the bigint type. It indicates the specified decimal place to which the result is to be rounded off. For all other input types, an error is returned. If it is omitted, the number is rounded to the ones place. The default value is 0.

Returned value: double or decimal type. If number or decimal_places is NULL, NULL is returned.

 **Note** decimal_places can be negative. Negative numbers are counted from the decimal point to left and the decimal part is left out; if the value of decimal_places is greater than the length of the integer part, 0 is returned.

Example:

```
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
```

6.7.1.17. SIN

Function declaration:

```
double sin(double number)
decimal sin(decimal number)
```

Purpose: It is used to return the sine of a number. The input must be a radian value.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double or decimal type. If number is NULL, NULL is returned.

6.7.1.18. SINH

Function declaration:

```
double sinh(double number)
decimal sinh(decimal number)
```

Purpose: It is used to return the hyperbolic sine of a number.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned.

Returned value: double or decimal type. If number is NULL, NULL is returned.

6.7.1.19. SQRT

Function declaration:

```
double sqrt(double number)
decimal sqrt(decimal number)
```

Purpose: It is used to return the square root of a number.

Description:

number: double or decimal type. It must be greater than 0. If it is less than 0, an error is returned. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other types of inputs, an error is returned.

Returned value: double or decimal type. If number is NULL, NULL is returned.

6.7.1.20. TAN

Function declaration:

```
double tan(double number)
decimal tan(decimal number)
```

Purpose: It is used to return the tangent of a number. The input must be a radian value.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other types of inputs, an error is returned.

Returned value: double or decimal type. If number is NULL, NULL is returned.

6.7.1.21. TANH

Function declaration:

```
double tanh(double number)
decimal tanh(decimal number)
```

Purpose: It is used to return the hyperbolic tangent of a number.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other types of inputs, an error is returned.

Returned value: double or decimal type. If number is NULL, NULL is returned.

6.7.1.22. TRUNC

Function declaration:

```
double trunc(double number[, bigint decimal_places])
decimal trunc(decimal number[, bigint decimal_places])
```

Purpose: It is used to truncate 'number' to the specified decimal place.

Description:

- number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted

into a value of the double type before this computation. For all other types of inputs, an error is returned.

- **decimal_places**: a constant of the bigint type. It indicates the decimal place to which a number is to be truncated. Numbers of other types are implicitly converted into values of the bigint type. If it is omitted, the result is truncated to the ones place by default.

Returned value: double or decimal type. If number or decimal_places is NULL, NULL is returned.

Note

- The truncated part is supplemented with 0.
- decimal_places can be negative. Negative numbers are truncated from the decimal point to the left and the decimal part is left out. If the value of decimal_places is greater than the length of the integer part, 0 is returned.

Example:

```
trunc(125.815) = 125.0
trunc(125.815, 0) = 125.0
trunc(125.815, 1) = 125.800000000000001
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
```

6.7.1.23. Additional mathematical functions

MaxCompute 2.0 provides additional mathematical functions. You must add the following SET statement before SQL statements contained in the UNHEX function:

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

 **Note** You must submit and execute the SET statement and the SQL statements of the new functions simultaneously.

The mathematical functions described in subsequent topics are new in MaxCompute 2.0.

6.7.1.24. LOG2

Function declaration:

```
Double log2(DOUBLE number)
Double log2(DECIMAL number)
```

Purpose: It is used to return the logarithm of number to base 2.

Description:

number: double or decimal type.

Returned value: double type. If the input is 0 or NULL, NULL is returned.

Example:

```
log2(null) = null
log2(0) = null
log2(8) = 3.0
```

6.7.1.25. LOG10

Function declaration:

```
Double log10(Double number)
Double log10(Decimal number)
```

Purpose: It is used to return the logarithm of number to base 10.

Description:

number: double or decimal type.

Returned value: double type. If the input is 0 or NULL, NULL is returned.

Example:

```
log10(null) = null
log10(0) = null
log10(8) = 0.9030899869919435
log10('abc') = null
```

6.7.1.26. BIN

Command syntax:

```
string bin(bigint number)
```

Purpose: It is used to return the binary format of a number.

Description:

number: bigint.

Returned value: string type. If the input is 0, 0 is returned. If the input is NULL, NULL is returned.

Example:

```
bin(0) = '0'
bin(null) = 'null'
bin(12) = '1100'
```

6.7.1.27. HEX

Function declaration:

```
STRING hex(BIGINT number)
STRING hex(STRING number)
STRING hex(BINARY number)
```

Purpose: It is used to convert an integer or character into hexadecimal format.

Description:

number: If this value is of the bigint type, the hexadecimal format of the number is returned. If this value is of the string type, the hexadecimal value of the string is returned.

Returned value: string type. If the input is 0, 0 is returned. If the input is NULL, NULL is returned.

Example:

```
hex(0) = '0'
hex('abc') = '616263'
hex(17) = '11'
hex('17') = '3137'
hex(null) = 'null'
```

6.7.1.28. UNHEX

Function declaration:

```
BINARY unhex(STRING number)
```

Purpose: It is used to return the regular character string represented in the hexadecimal format.

Description:

number: a hexadecimal string.

Returned value: binary type. If the input is 0, a failure is returned. If the input is NULL, NULL is returned.

Example:

```
unhex('616263') = 'abc'
unhex(616263) = 'abc'
```

6.7.1.29. RADIANS

Command syntax:

```
double radians(double number)
```

Purpose: It is used to convert degrees into radians.

Description:

number: double type

Returned value: double type. If the input is NULL, NULL is returned.

Example:

```
radians(90) = 1.5707963267948966  
radians(0) = 0.0  
radians(null) = null
```

6.7.1.30. DEGREES

Function declaration:

```
DOUBLE degrees(DOUBLE number)  
DOUBLE degrees(DECIMAL number)
```

Purpose: It is used to convert radians into degrees.

Description:

number: double or decimal type.

Returned value: double type. If the input is NULL, NULL is returned.

Example:

```
degrees(1.5707963267948966) = 90.0  
degrees(0) = 0.0  
degrees(null) = null
```

6.7.1.31. SIGN

Function declaration:

```
DOUBLE sign(DOUBLE number)  
DOUBLE sign(DECIMAL number)
```

Purpose: It is used to indicate the sign of the input data. 1.0 indicates positive and -1.0 indicates negative. 0.0 indicates 0.

Description:

number: double or decimal type.

Returned value: double type. If the input is 0, 0.0 is returned. If the input is NULL, NULL is returned.

Example:

```
sign(-2.5) = -1.0  
sign(2.5) = 1.0  
sign(0) = 0.0  
sign(null) = null
```

6.7.1.32. E

Function declaration:

```
DOUBLE e()
```

Purpose: It is used to return the value of e (Euler's number).

Returned value: double type.

Example:

```
e() = 2.718281828459045
```

6.7.1.33. PI

Function declaration:

```
DOUBLE pi()
```

Purpose: It is used to return the value of π .

Returned value: double type.

Example:

```
pi() = 3.141592653589793
```

6.7.1.34. FACTORIAL

Function declaration:

```
BIGINT factorial(INT number)
```

Purpose: It is used to return the factorial of number.

Description:

number: int type. Value range: 0 to 20.

Returned value: bigint type. If the input is 0, 1 is returned. If the input is NULL or any value outside the range of 0 to 20, NULL is returned.

Example:

```
factorial(5) = 120 --5! = 5*4*3*2*1 = 120
```

6.7.1.35. CBRT

Command syntax:

```
double cbrt(double number)
```

Purpose: It is used to return the cube root of a number.

Description:

number: double type.

Returned value: double type. If the input is NULL, NULL is returned.

Example:

```
cbrt(8) = 2
cbrt(null) = null
```

6.7.1.36. SHIFTLLEFT

Function declaration:

```
INT shiftright(TINYINT|SMALLINT|INT number1, INT number2)
BIGINT shiftright(BIGINT number1, INT number2)
```

Purpose: It is used to shift left a value by a given number of places (<<).

Description:

- number1: an integer of the tinyint, smallint, int, or bigint type.
- number2: an integer of the int type.

Returned value: int or bigint type.

Example:

```
shiftright(1,2) = 4
-- Shift left the binary value of 1 by two places (1<<2, 0001 changed to 0100)
shiftright(4,3) = 32
-- Shift left the binary value of 4 by three places (4<<3, 0100 changed to 100000)
```

6.7.1.37. SHIFTRIGHT

Function declaration:

```
INT shiftright(TINYINT|SMALLINT|INT number1, INT number2)
BIGINT shiftright(BIGINT number1, INT number2)
```

Purpose: It is used to shift right a value by a given number of places (>>).

Description:

- number1: an integer of the tinyint, smallint, int, or bigint type.
- number2: an integer of the int type.

Returned value: int or bigint type.

Example:

```
shiftright(4,2) = 1
-- Shift right the unsigned binary value of 4 by two places (4>>2, 0100 changed to 0001)
shiftright(32,3) = 4
-- Shift right the unsigned binary value of 32 by two places (32>>3, 100000 changed to 0100 )
```

6.7.1.38. SHIFTRIGHTUNSIGNED

Function declaration:

```
INT shiftrightunsigned(TINYINT|SMALLINT|INT number1, INT number2)
BIGINT shiftrightunsigned(BIGINT number1, INT number2)
```

Purpose: It is used to shift right an unsigned value by a given number of places (>>>).

Description:

- number1: an integer of the tinyint, smallint, int, or bigint type.
- number2: an integer of the int type.

Returned value: int or bigint type.

Example:

```
shiftrightunsigned(8,2) = 2
-- In this example, shift right the unsigned binary value of 8 (1000 in binary) by two places and return 2 (0010 in binary).
shiftrightunsigned(-14,2) = 1073741820
-- Shift right the unsigned binary value of -14 by two places (-14>>>2, 11111111 11111111 11111111 11110010 changed to 00111111 11111111 11111111 11111100)
```

6.7.2. String processing functions

6.7.2.1. CHAR_MATCHCOUNT

Command syntax:

```
bigint char_matchcount(string str1, string str2)
```

Purpose: It is used to return the number of characters in str1 that appear in str2 (repeated characters are not counted).

Description:

str1 and str2: string type. Both must be valid UTF-8 strings. If invalid characters are found during matching, a negative value is returned.

Returned value: bigint type. If any input is NULL, NULL is returned.

Example:

```
char_matchcount('abd', 'aabc') = 2
-- The a and b characters in str1 appear in str2.
```

6.7.2.2. CHR

Command syntax:

```
string chr(bigint ascii)
```

Purpose: It is used to convert an ASCII code into the corresponding character.

Description:

ascii: ASCII value of the bigint type. If the input is of the string, double, or decimal type, it is implicitly converted into a value of the bigint type before this computation. If the input is of another type, an error is returned.

Returned value: string type. The parameter value range is from 0 to 255. A value out of range will cause an error. If the input is NULL, NULL is returned.

6.7.2.3. CONCAT

Command syntax:

```
string concat(string a, string b...)
```

Purpose: It is used to join input strings into a single string.

Description:

a, b...: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type. For all other input types, an error is returned.

Returned value: string type. If there is no input or if any input is NULL, NULL is returned.

Example:

```
concat('ab', 'c') = 'abc'
concat() = null
concat('a', null, 'b') = null
```

6.7.2.4. INSTR

Function declaration:

```
bigint instr(string str1, string str2[, bigint start_position[, bigint nth_appearance]])
```

Purpose: It is used to calculate the position of substring str2 in string str1.

Description:

- str1: string type. It indicates a string to be searched. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.
- str2: string type. It indicates a substring to be searched out. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.
- start_position: bigint type. If it is of another type, an error is returned. It indicates which character in str1 the search will start with. The default start position is the first character, marked as 1.
- nth_appearance: bigint type. If it is greater than 0, it indicates the position where the substring matches the string for the nth_appearance time. If it is of another type or if it is less than or equal to 0, an error is returned.

Returned value: bigint type.

 Note

- If str2 is not found in str1, 0 is returned.
- If any input is NULL, NULL is returned.
- If str2 is NULL, the matching will always be successful. Therefore, 1 is returned for instr('abc', '').

Example:

```
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
```

6.7.2.5. IS_ENCODING

Function declaration:

```
boolean is_encoding(string str, string from_encoding, string to_encoding)
```

Purpose: It is used to determine whether an input string can be converted from a specified character set (from_encoding) to another character set (to_encoding). It can be used to determine whether the input is garbled. from_encoding is usually set to utf-8, and to_encoding is set to gbk.

Description:

- str: string type. If the input is NULL, NULL is returned. Null is considered to belong to any character set.
- from_encoding, to_encoding: string type. They indicate the source and the destination character sets respectively. If the input is NULL, NULL is returned.

Returned value: boolean type. If a string is converted successfully, true is returned. Otherwise, false is returned.

Example:

```
is_encoding('test', 'utf-8', 'gbk') = true
is_encoding('test', 'utf-8', 'gbk') = true
-- These two traditional Chinese characters are in GBK stock in China.
is_encoding('test', 'utf-8', 'gb2312') = false
-- The grapheme inventory of 'GB2312' does not contain these two Chinese characters.
```

6.7.2.6. KEYVALUE

Function declaration:

```
KEYVALUE(String srcStr, String split1, String split2, String key)
KEYVALUE(String srcStr, String key) //split1 = ";", split2 = ":"
```

Purpose: It is used to split the source string into key-value pairs by split1, separate key-value pairs by split2, and return the value of the corresponding key.

Description:

- **srcStr**: the source string to be split.
- **key**: string type. After the source string is split by 'split1' and 'split2', return the corresponding value according to the specification of the 'key' value.
- **split1** and **split2**: strings used as separators. The source string is split by the two separators. If these two parameters are not specified in the expression, split1 is a semicolon (;) and split2 is a colon (:) by default. If a string that has been split by split1 has multiple split2 values, the returned result is undefined.

Returned value: string type.

- If 'split1' or 'split2' is NULL, return NULL.
- If 'srcStr' and 'key' are NULL or if there is no matched 'key', return NULL.
- If multiple 'key-value' matches, return the value corresponding to the first matched key.

Example:

```
keyvalue('0:1\;1:2', 1) = '2'
-- The source string is "0:1\;1:2". Because split1 and split2 are not specified, split1 is
a semicolon (;) and split2 is a colon (:) by default. After split1 split, the key-value pair
is:
0:1\,1:2
After split2 split, it becomes:
0 1/
1 2
Returns the value(2) of the key corresponding to 1.
keyvalue("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;", "\;",";", "tf") = "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 split1 "\;", the
key-value pairs are as follows:
decreaseStore:1, xcard:1, isB2C:1, tf:21910, cart:1, shipping:2, pf:0, market:shoes, instPayAmount:0
If split2 is ":", after split it becomes:
decreaseStore 1
xcard 1
isB2C 1
tf 21910
cart 1
shipping 2
pf 0
market shoes
instPayAmount 0
For the key parameter whose value is "tf", the returned value of the corresponding value parameter is 21910.
```

6.7.2.7. LENGTH

Function declaration:

```
bigint length(string str)
```

Purpose: It is used to return the length of a string.

Description:

str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.

Returned value: bigint type. If a string is NULL, NULL is returned. If a string is not UTF-8 encoded, -1 is returned.

Example:

```
length('hi! China') = 6
```

6.7.2.8. LENGTHB

Function declaration:

```
bigint lengthb(string str)
```

Purpose: It is used to return the length of a string. Unit: byte.

Description:

str: string type. If the input is of the bigint, double, decimal, or datetime type, it is implicitly converted into a value of the string type before this computation. If the input is of another type, an error is returned.

Returned value: bigint type. If the input is NULL, NULL is returned.

Example:

```
lengthb('hi! china') = 10
```

6.7.2.9. MD5

Function declaration:

```
string md5(string value)
```

Purpose: It is used to calculate the MD5 value of the input string value.

Description:

value: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. If the input is of another type, an error is returned.

Returned value: string type. If the input is NULL, NULL is returned.

6.7.2.10. PARSE_URL

Function declaration:

```
STRING PARSE_URL(STRING url, STRING part[,STRING key])
```

Purpose: It is used to parse a URL and extract information by key.

Description:

- If URL or part is NULL, NULL is returned. If URL is invalid, an error is returned.
- part: string type. It supports HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO, and is case insensitive. If it is none of the preceding values, an error is returned.
- If part is QUERY, the value in query string that corresponds to the key value is extracted. Otherwise, the parameter key is ignored.

Returned value: string type.

Example:

```
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"
```

6.7.2.11. REGEXP_EXTRACT

Command syntax:

```
string regexp_extract(string source, string pattern[, bigint occurrence])
```

Purpose: It is used to return part of the source string that matches the regular expression and the occurrence of the matches.

Description:

- source: string type. It indicates a string to be searched.
- pattern: string type. If pattern is NULL or if there is no specified group in pattern, an error is returned.
- occurrence: bigint type. It must be a number that is greater than or equal to 0. Otherwise, an error is returned. The default value is 1 if it is not specified. If it is 0, a substring which meets all pattern requirements is returned.

Returned value: string type. If any input is NULL, NULL is returned.

Example:

```

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 at the MaxCompute client, two backslashes (\\) are
needed to be used as the escape character.
regexp_extract('foothebar', 'foothebar')
-- An error is returned because no part is specified in the pattern.

```

6.7.2.12. REGEXP_INSTR

Function declaration:

```

bigint regexp_instr(string source, string pattern[, bigint start_position[, bigint nth_occurrence[, bigint return_option]])

```

Purpose: It is used to return the start or end position of the substring that matches the pattern in the source string from start_position for the nth_occurrence time.

Description:

- **source:** string type. It indicates a string to be searched.
- **pattern:** a constant of the string type. If pattern is null, an error is returned.
- **start_position:** a constant of 'bigint' type. It is the start position for the search. When it is not specified, it is 1 by default. If it is of another type or less than or equal to 0, an error is returned.
- **nth_occurrence:** a constant of the bigint type. When it is not specified, it is 1 by default, indicating the position where a substring matches pattern in search for the first time. If it is of another type or if it is less than or equal to 0, an error is returned.
- **return_option:** a constant of the bigint type. The value is either 0 or 1. If it is of another type or the value is not supported, an error is returned. 0 indicates that the start position of the matched substring is returned, and 1 indicates that the end position of the matched substring is returned.

Returned value: bigint type. It is the start or end position of the matched substring in source string according to the type specified by return_option. If any input is NULL, NULL is returned.

Example:

```

regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14

```

6.7.2.13. REGEXP_SUBSTR

Function declaration:

```

string regexp_substr(string source, string pattern[, bigint start_position[, bigint nth_occurrence]])

```

Purpose: It is used to return the string that matches pattern in the source string from position start_position for the nth_occurrence time.

Description:

- **source:** string type. It indicates a string to be searched.

- **pattern**: a constant of the string type. It indicates a pattern to be matched. If pattern is null, an error is returned.
- **start_position**: a constant of the bigint type. It must be greater than 0. If it is another type or if it is less than or equal to 0, an error is reported. When it is not specified, it is regarded as 1 by default, so the matching starts from the first character of 'source'.
- **nth_occurrence**: a constant of the bigint type. It must be greater than 0. If it is another type or is less than or equal to 0, an error is returned. If it is not specified, it is regarded as 1 by default, indicating that the string in the first match is returned.

Returned value: string type. If any input is NULL, NULL is returned. If there is no matching, NULL is returned.

Example:

```
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"
```

6.7.2.14. REGEXP_COUNT

Command syntax:

```
bigint regexp_count(string source, string pattern[, bigint start_position])
```

Purpose: It is used to return the number of occurrences that a string pattern appears in the source string, starting from start_position.

Description:

- **source**: string type. It indicates a string to be searched. For all other input types, an error is returned.
- **pattern**: string type. It indicates a pattern to be matched. If the pattern is NULL or of another type, an error is returned.
- **start_position**: bigint start_position must be a number that is greater than 0. Otherwise, an error is returned. If start_position is not specified, the default value is 1 which means starting from the first character of the source string.

Returned value: bigint type. If any input is NULL, NULL is returned. If there is no matching, 0 is returned.

Example:

```
regexp_count('abababc', 'a.c') = 1
regexp_count('abcde', '[[[:alpha:]]{2}%', 3) = 1
```

6.7.2.15. SPLIT_PART

Function declaration:

```
string split_part(string str, string delimiter, bigint start[, bigint end])
```

Purpose: It is used to split a string with the specified delimiter, and return the string between the specified start segment and end segment (inclusive).

Description:

- **str**: string type. It indicates a string to be split. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. If the input is of any other type, an error is returned.
- **delimiter**: a constant of the string type. It indicates the delimiter used to split a string. It can be a character or a string. If it is neither a character nor a string, an error is returned.
- **start**: a constant of the bigint type. It must be greater than 0. If it is not a constant or is of a different type, an error is returned. It indicates the start number (starting from 1) of the segment to be returned. 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 it is not a constant or is of a different type, an error is returned. If end is not specified, the last segment is returned.

Returned value: string type. If any input is NULL, NULL is returned. If delimiter is NULL, the original string is returned.

 Note

- If delimiter does not exist in str, and start is set to 1, the entire str is returned. If the input is NULL, NULL is returned.
- If start is set to a value greater than the number of segments (for example, the string has 6 segments but the start value is greater than 6), NULL is returned.
- If end is set to a value greater than the number of segments, the string between start and the last segment is returned.

Example:

```
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) = ''
```

6.7.2.16. REGEXP_REPLACE

Function declaration:

```
string regexp_replace(string source, string pattern, string replace_string[, bigint occurrence])
```

Purpose: It is used to search a source string for substrings that match a given pattern, replace them with the specified replace_string, and return the result.

Description:

- **source**: string type. It indicates a string to be replaced.
- **pattern**: a constant of the string type. It indicates a pattern to be matched. If pattern is null, an error is returned.
- **replace_string**: string type. It is used to replace the matched pattern.
- **occurrence**: a constant of the bigint type. It must be greater than or equal to 0. This parameter indicates the number of times at which the substring matches the pattern for replacement with replace_string. If the input value is 0, all matched substrings are replaced. If it is of another type or

less than 0, an error is returned. It can be omitted. The default value is 0.

Returned value: string type. When the referenced group does not exist, the replace operation is not performed. When the input parameters source, pattern, and occurrence are NULL, NULL is returned. If replace_string is NULL and the pattern is matched, NULL is returned. If replace_string is NULL but the pattern is not matched, the original string is returned.

 **Note** When the referenced group does not exist, the action is not defined.

Example:

```
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 a group is defined in pattern and the referenced second group is not existent.
-- Please avoid this. The result to reference nonexistent group is not defined.
regexp_replace("abcd", "(. *) (. )$", "\\2", 0) = "d"
regexp_replace("abcd", "a", "\\1", 0) = "bcd"
-- No group definition is in pattern, so '\\1' references a nonexistent group.
-- Try to avoid this. The result of referencing a nonexistent group is not defined.
```

6.7.2.17. SUBSTR

Function declaration:

```
string substr(string str, bigint start_position[, bigint length])
```

Purpose: It is used to return a substring of 'length' from 'str' starting from 'start_position'.

Description:

- str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other types of inputs, an error is returned.
- start_position: bigint type. The start position is 1. If start_position is a negative value, the counting starts from the end to the start of the string and the last character is -1. If the input is of another type, an error is returned.
- length: bigint type. It indicates the length of the substring, which is greater than 0. If it is of another type or less than or equal to 0, an error is returned.

Returned value: string type. If any input is NULL, NULL is returned.

 **Note** If the length is omitted, the substring from start to end is returned.

Example:

```
substr("abc", 2) = "bc"
substr("abc", 2, 1) = "b"
substr("abc", -2, 2) = "bc"
substr("abc", -3) = "abc"
```

6.7.2.18. TOLOWER

Function declaration:

```
string tolower(string source)
```

Purpose: It is used to convert 'source' into a lowercase string and return the value.

Description:

source: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
tolower("aBcd") = "abcd"  
tolower("Haha Cd") = "haha cd"
```

6.7.2.19. TOUPPER

Function declaration:

```
string toupper(string source)
```

Purpose: It is used to convert 'source' into an uppercase string and return the value.

Description:

source: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other types of inputs, an error is returned.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
toupper("aBcd") = "ABCD"  
toupper("HahaCd") = "HAHACD"
```

6.7.2.20. TO_CHAR

Function declaration:

```
string to_char(boolean value)  
string to_char(bigint value)  
string to_char(double value)  
string to_char(decimal value)
```

Purpose: It is used to convert the input of the boolean, bigint, decimal, or double type into a value of the string type.

Description:

value: boolean, bigint, or double type. For all other types of inputs, an error is returned. For more information about the formatted output of data of the datetime type, see [Date processing functions — TO_CHAR](#).

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
to_char(123) = '123'
to_char(true) = 'TRUE'
to_char(1.23) = '1.23'
to_char(null) = 'null'
```

6.7.2.21. TRIM

Function declaration:

```
string trim(string str)
```

Purpose: It is used to remove the spaces from both ends of 'str'.

Description:

str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other types of inputs, an error is returned.

Returned value: string type. If the input is NULL, NULL is returned.

6.7.2.22. LTRIM

Function declaration:

```
string ltrim(string str)
```

Purpose: It is used to remove the left spaces for input string str.

Description:

str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
select ltrim(' abc ') from dual;
-- Returned result:
+-----+
| _c0 |
+-----+
| abc |
+-----+
```

6.7.2.23. RTRIM

Function declaration:

```
string rtrim(string str)
```

Purpose: It is used to remove the rightmost spaces from the input string 'str'.

Description:

str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
select rtrim('a abc ') from dual;
-- Returned result:
+-----+
| _c0 |
+-----+
| a abc |
+-----+
```

6.7.2.24. REVERSE

Function declaration:

```
STRING REVERSE(string str)
```

Purpose: It is used to return a reverse string.

Description:

str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
select reverse('abcdefg') from dual;
-- Returned result:
+-----+
| _c0 |
+-----+
| gfdecba |
+-----+
```

6.7.2.25. SPACE

Function declaration:

```
STRING SPACE(bigint n)
```

Purpose: It is used to return a string with 'n' consecutive space characters.

Description:

n: bigint type. The length cannot exceed 2 MB. If the input is NULL, an error is returned.

Returned value: string type.

Example:

```
select length(space(10)) from dual;
-- 10 is returned.
select space(4000000000000) from dual;
-- An error is returned as the length exceeds 2 MB.
```

6.7.2.26. REPEAT

Function declaration:

```
STRING REPEAT(string str, bigint n)
```

Purpose: It is used to return string 'str' that has been repeated n times.

Description:

- str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.
- n: bigint type. The length cannot exceed 2 MB. If it is NULL, an error is returned.

Returned value: string type.

Example:

```
select repeat('abc',5) from lxw_dual;
-- abcabcabcabcabc is returned.
```

6.7.2.27. ASCII

Function declaration:

```
Bigint ASCII(string str)
```

Purpose: It is used to return the ASCII code of the first character of string 'str'.

Description:

str: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.

Returned value: bigint type.

Example:

```
select ascii('abcde') from dual;  
-- 97 is returned.
```

6.7.2.28. URL_ENCODE

Function declaration:

```
STRING URL_ENCODE(STRING input[, STRING encoding])
```

Purpose: It is used to encode the input string in the application/x-www-form-urlencoded MIME format:

- a-z and A-Z remain unchanged.
- ".", "-", "*", and "_" remain unchanged.
- Spaces are converted into "+".
- The rest of the characters are converted into byte values according to the specified encoding. If encoding is not specified, UTF-8 is used by default. In this case, each byte value is represented in the %xy format, where xy represents the hexadecimal form of the character.

Description:

- input: string type.
- encoding: specifies an encoding format. If it is not specified, UTF-8 is used by default.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
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"
```

6.7.2.29. URL_DECODE

Function declaration:

```
STRING URL_DECODE(STRING input[, STRING encoding])
```

Purpose: It is used to convert an input string from the application/x-www-form-urlencoded MIME format into a normal string. This is the inverse function of URL_ENCODE:

- a-z and A-Z remain unchanged.
- ".", "-", "*", and "_" remain unchanged.
- "+" is converted into a space.
- The %xy formatted sequence is converted into byte values. Consecutive byte values are interpreted as the corresponding strings based on the input encoding.
- Other characters remain unchanged.
- The final returned value of the function is a UTF-8 string.

Description:

- input: string type.
- encoding: specifies an encoding format. If it is not specified, UTF-8 is used by default.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
url_decode('%E7%A4%BA%E4%BE%8Bfor+url_encode%3A%2F%2F+%28fdsf%29')= "Example for url_encode :// (fdsf)"
url_decode('Exaple+for+url_encode+%3A%2F%2F+dsf%28fasfs%29', 'GBK') = "Exaple for url_encode :// dsf(fasfs)" ````
```

6.7.2.30. Additional string processing functions

MaxCompute 2.0 provides additional string processing functions. You must add the following SET statement before SQL statements contained in the LPAD, RPAD, and TRANSLATE functions:

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

 **Note** You must submit and execute the SET statement and the SQL statements of the new functions simultaneously.

The string processing functions described in subsequent topics are new in MaxCompute 2.0.

6.7.2.31. CONCAT_WS

Command syntax:

```
string concat_ws(string SEP, string a, string b...)
```

Purpose: It is used to join input strings starting from the second with the first string as the separator.

Description:

- SEP: delimiter of the string type. If it is not specified, an error is returned.
- a, b...: string type. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type. For all other input types, an error is returned.

Returned value: string type. If there is no input or if any input is NULL, NULL is returned.

Example:

```
concat_ws(':', 'name', 'bob') = 'name:bob'
concat_ws(':', 'avg', null, '34') = 'null'
```

6.7.2.32. LPAD

Function declaration:

```
string lpad(string a, int len, string b)
```

Purpose: It is used to pad the left side of string a with string b until the new padded string has len bits.

Description:

- len: int type.
- a, b: string type.

Returned value: string type. If len is less than the number of bits in a, a is truncated from the left to obtain a string with the number of bits specified by len. If len is 0, NULL is returned.

Example:

```
lpad('abcdefgh',10,'12')='12abcdefgh'
lpad('abcdefgh',5,'12')='abcde'
lpad('abcdefgh',0,'12')
-- NULL is returned.
```

6.7.2.33. RPAD

Function declaration:

```
string rpad(string a, int len, string b)
```

Purpose: It is used to pad the right side of string 'a' with string 'b' until the new padded string has 'len' places.

Description:

- len: int type.
- a, b: string type.

Returned value: string type. 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, NULL is returned.

Example:

```
rpadd('abcdefgh',10,'12')='abcdefgh12'
rpadd('abcdefgh',5,'12')='abcde'
rpadd('abcdefgh',0,'12')
-- NULL is returned.
```

6.7.2.34. REPLACE

Function declaration:

```
string replace(string a, string OLD, string NEW)
```

Purpose: It is used to replace the part of string a that is exactly the same as string OLD with string NEW, and return string a.

Description:

All parameters are of the string type.

Returned value: string type. If any input is NULL, NULL is returned.

Example:

```
replace('ababab','abab','12') = '12ab'
replace('ababab','cdf','123') = 'ababab'
replace('123abab456ab',null,'abab') = 'null'
```

6.7.2.35. SOUNDEX

Function declaration:

```
string soundex(string a)
```

Purpose: It is used to convert an ordinary string into a soundex string.

Description:

All parameters are of the string type.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
soundex('hello') = 'H400'
```

6.7.2.36. SUBSTRING_INDEX

Function declaration:

```
string substring_index(string a, string SEP, int count))
```

Purpose: It is used to return the substring in 'a' that comes before the 'count' (nth) delimiter ('SEP'). If 'count' is a positive value, it starts from the left of the string. If 'count' is a negative value, it starts from the right of the string.

Description:

- a, SEP: string type.
- count: int type.

Returned value: string type. If the input is NULL, NULL is returned.

Example:

```
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'
```

6.7.2.37. TRANSLATE

Function declaration:

```
string translate(string|varchar str1, string|varchar str2, string|varchar str3)
```

Purpose: It is used to replace str2 in str1 with str3.

Returned value: STRING type. If any input is NULL, NULL is returned.

Example:

```
translate('MaxComputer', 'puter', 'pute')='MaxCompute'  
translate('aaa', 'b', 'c')='aaa'  
translate('MaxComputer', 'puter', null)=null
```

6.7.2.38. JSON_TUPLE

Function declaration:

```
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, which indicates a standard JSON string.
- key: a value of the STRING type, which is used to describe 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.

Note

- 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 parameter does not exist in the JSON string, it is considered invalid.
- If the json parameter is valid and the key parameter exists, the required string is returned.
- This function parses a JSON string 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, use JSON_TUPLE with LATERAL VIEW.

Example:

The school table contains the following data:

```

Table: school
+-----+-----+
| Id      | json      |
+-----+-----+
| 1       | {
          "School name": "湖畔大学",
          "Location": "杭州",
          "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;
-- Equivalent to the following statement: 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 |
+-----+-----+
| 湖畔大学 | 杭州 |
+-----+-----+

```

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 |
+-----+-----+-----+
```

6.7.3. Date processing functions

6.7.3.1. DATEADD

Function declaration:

```
datetime dateadd(datetime date, bigint delta, string datepart)
```

Purpose: It is used to modify date based on delta and datepart.

Description:

- **date:** This value must be a string type date. If the input is of the string type, it is implicitly converted into a value of the datetime type before this computation. For all other types of inputs, an error is returned.
- **delta:** bigint type. It indicates the scope of modification. If the input is of the string or double type, it is implicitly converted into a value of the bigint type before this computation. If the input is of another type, an error is returned. If delta is greater than 0, the delta is added to the value. If delta is less than 0, the delta is subtracted from the value.
- **datepart:** a constant of the string type. This field is set based on the string-datetime conversion convention. yyyy indicates year and mm indicates month. For rules of type conversion, see [Conversion between the string type and datetime type](#). In addition, the extended date format is also supported: year, month or mon, day, and hour. If the parameter value is not a constant or of an unsupported format or another type, an error is returned.

Returned value: datetime type. If any input is NULL, NULL is returned.

Note

- When delta is added to or subtracted from the value, carrying and borrowing are base-10 for year, base-12 for month, base-24 for hour, and base-60 for minute and second. If delta is measured in months, the following calculation is applied: If the month in the datetime value does not cause the day value to become invalid after delta is added, the day value is kept. Otherwise, the day value is adjusted to the last day of the resulting month.
- This field is set based on the string-datetime conversion convention. yyyy indicates the year and mm indicates the month. Unless otherwise specified, all built-in functions related to the datetime type follow this convention. Unless otherwise specified, the datepart of all built-in functions related to the datetime type also supports the extended date format: year, month or mon, day, and hour.

Example:

```
If trans_date = 2017-02-28 00:00:00:
dateadd(trans_date, 1, 'dd') = 2017-03-01 00:00:00
-- Add one day. The result is beyond the last day of February. The actual value is the first
day of next month.
dateadd(trans_date, -1, 'dd') = 2017-02-27 00:00:00
-- Subtract one day.
dateadd(trans_date, 20, 'mm') = 2018-10-28 00:00:00
-- 20 months are added. The month overflows, and 1 is added to the year.
trans_date = 2017-02-28 00:00:00, dateadd(transdate, 1, 'mm') = 2017-03-28 00:00:00
trans_date = 2017-01-29 00:00:00, dateadd(transdate, 1, 'mm') = 2017-02-28 00:00:00
-- February has 28 days only, so the last day of the month is returned.
trans_date = 2017-03-30 00:00:00, dateadd(transdate, -1, 'mm') = 2017-02-28 00:00:00
```

The values of trans_date used only serve as examples. The datetime examples in this document use simple formats. In MaxCompute SQL, a constant cannot be of the datetime type. The following syntax is incorrect:

```
select dateadd(2017-03-30 00:00:00, -1, 'mm') from tbl1;
```

If you must use a constant of the datetime type, use the following method:

```
select dateadd(cast("2017-03-30 00:00:00" as datetime), -1, 'mm') from tbl1;
-- The String type constant is converted to datetime type by explicit conversion.
```

6.7.3.2. DATEDIFF

Function declaration:

```
bigint datediff(datetime date1, datetime date2, string datepart)
```

Purpose: It is used to calculate the difference between date1 and date2 based on the specified datepart.

Description:

- date1 and date2: minuend and subtrahend of the datetime type respectively. If the input is a string,

it is implicitly converted into a value of the datetime type before this computation. For all other input types, an error is returned.

- **datepart**: A constant of the string type. It supports the extended date format. If datepart is not in the specified format or is of another type, an error is returned.

Returned value: bigint type. If any input is NULL, NULL is returned. If date1 is less than date2, the returned value may be negative.

 **Note** The lower unit part is truncated based on 'datepart' in the computation process and then the result is calculated.

Example:

```
If start = 2017-12-31 23:59:59 and end = 2018-01-01 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('2017-05-31 13:00:00', '2017-05-31 12:30:00', 'ss') = 1800
datediff('2017-05-31 13:00:00', '2017-05-31 12:30:00', 'mi') = 30
```

6.7.3.3. DATEPART

Function declaration:

```
bigint datepart(datetime date, string datepart)
```

Purpose: It is used to extract the value of the specified datepart in date.

Description:

- **date**: datetime type. If the input is a string, it is implicitly converted into a value of the datetime type before this computation. For all other input types, an error is returned.
- **datepart**: a constant of the string type. It supports the extended date format. If datepart is not in the specified format or is of another type, an error is returned.

Returned value: bigint type. If any input is NULL, NULL is returned.

Example:

```
datepart('2017-06-08 01:10:00', 'yyyy') = 2017
datepart('2017-06-08 01:10:00', 'mm') = 6
```

6.7.3.4. DATETRUNC

Function declaration:

```
datetime datetrunc (datetime date, string datepart)
```

Purpose: It is used to return the value of a date after the specified datepart is truncated.

Description:

- **date:** datetime type. If the input is a string, it is implicitly converted into a value of the datetime type before this computation. For all other input types, an error is returned.
- **datepart:** a constant of the string type. It supports the extended date format. If datepart is not in the specified format or is of another type, an error is returned.

Returned value: datetime type. If any input is NULL, NULL is returned.

Example:

```
datetrunc('2017-12-07 16:28:46', 'yyyy') = 2017-01-01 00:00:00
datetrunc('2017-12-07 16:28:46', 'month') = 2017-12-01 00:00:00
datetrunc('2017-12-07 16:28:46', 'DD') = 2017-12-07 00:00:00
```

6.7.3.5. GETDATE

Function declaration:

```
datetime getdate()
```

Purpose: It is used to obtain the current system time. Use UTC+8 as the standard time of MaxCompute.

Returned value: the current date and time of the datetime type.

 **Note** In a MaxCompute SQL task (executed in a distributed manner), 'getdate' always returns a fixed value. The returned result is any time in MaxCompute. The time returned is precise to the second. In later versions, the time will be precise to the millisecond.

6.7.3.6. ISDATE

Function declaration:

```
boolean isdate(string date, string format)
```

Purpose: It is used to determine whether a date string can be converted into a date value based on the corresponding format string. If the conversion can be performed, true is returned. Otherwise, false is returned.

Description:

- **date:** This value must be a string type date. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other input types, an error is returned.
- **format:** a constant of the string type. The extended date format is not supported. If it is of another type or an unsupported format, an error is returned. If there are redundant format strings appearing in 'format', the date value corresponding to the first format string is used. Other strings are taken as delimiters. If isdate("1234-yyyy", "yyyy-yyyy"), true is returned.

Returned value: boolean type. If any input is NULL, NULL is returned.

6.7.3.7. LASTDAY

Function declaration:

```
datetime lastday(datetime date)
```

Purpose: It is used to return the last day of the current month to which the date belongs. The value is accurate to day. The hour, minute, and second part is expressed as 00:00:00.

Description:

date: datetime type. If the input is a string, it is implicitly converted into a value of the datetime type before this computation. For all other input types, an error is returned.

Returned value: datetime type. If the input is NULL, NULL is returned.

6.7.3.8. TO_DATE

Function declaration:

```
datetime to_date(string date, string format)
```

Purpose: It is used to convert the 'date' string into a date value.

Description:

- **date:** string type. It indicates the date value of the string type to be converted. If the input is of the bigint, decimal, double, or datetime type, it is implicitly converted into a value of the string type before this computation. For all other types of inputs or NULL, an error is returned.
- **format:** a constant of the string type in the date format. For all other types of inputs and non-constant values, an error is returned. It does not support the extended date format. Other characters are ignored as invalid characters in parsing. The format parameter must contain yyyy. Otherwise, an error is returned. If there are redundant format strings in the format, the corresponding date value of the first format string is used, and the rest are processed as separators. For example, `to_date('1234-2234', 'yyyy-yyyy')` returns '1234-01-01 00:00:00'.

Returned value: datetime type. The format is *yyyy-mm-dd hh:mi:ss*. If any input is NULL, NULL is returned.

Example:

```
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.
```

6.7.3.9. TO_CHAR

Function declaration:

```
string to_char(datetime date, string format)
```

Purpose: It is used to convert a value of the date type into a string based on the specified format.

Description:

- **date:** date value of the datetime type to be converted. If the input is a string, it is implicitly converted into a value of the datetime type before this computation. For all other types of inputs, an error is returned.
- **format:** a constant of the string type. If it is not a constant or is of a different type, an error is returned. In format, the date format part is replaced with the corresponding data and other characters are output directly.

Returned value: string type. If any input is NULL, NULL is returned.

Example:

```
to_char('2017-12-03 00:00:00', 'Alibabayyy-mm*dd') = 'Alibaba2017-12*03'
to_char('2017-07-18 00:00:00', 'yyyymmdd') = '20170718'
to_char('Alibaba 2017-12*3', 'Alibaba yyyy-mm*dd')
-- Null is returned.
to_char('2017-24-01', 'yyyy')
-- Null is returned.
to_char('2017718', 'yyyymmdd')
-- Null is returned.
```

 **Note** For more information about conversion from other types into the string type, see [String functions — TO_CHAR](#).

6.7.3.10. UNIX_TIMESTAMP

Function declaration:

```
bigint unix_timestamp(datetime date)
```

Purpose: It is used to convert a date into a datetime value of the integer type in the Unix format.

Description:

date: datetime type. It indicates the date. If the input is a string, it is implicitly converted into a value of the datetime type before this computation. For all other input types, an error is returned.

Returned value: bigint type. It indicates the date value in Unix format. If date is NULL, NULL is returned.

6.7.3.11. FROM_UNIXTIME

Function declaration:

```
datetime from_unixtime(bigint unixtime)
```

Purpose: It is used to convert a Unix date value from the BIGINT type to the DATETIME type.

Description:

unixtime: BIGINT type. It is a date value in the Unix format. If the input is of the STRING, DECIMAL, or DOUBLE type, it is implicitly converted into a value of the BIGINT type before computation.

Returned value: DATETIME type. If unixtime is NULL, NULL is returned.

 **Note** In the HIVE-compatible mode (where `set odps.sql.hive.compatible=true;` has been executed), if the input is of the STRING type, a date value of the STRING type is returned.

Example:

```
from_unixtime(123456789) = 1973-11-30 05:33:09;
```

6.7.3.12. WEEKDAY

Function declaration:

```
bigint weekday (datetime date)
```

Purpose: It is used to return the day of week for the specified date.

Description:

date: datetime type. If the input is of the string type, it is implicitly converted to a value of the datetime type before this computation. For all other input types, an error is returned.

Returned value: bigint type. If the input is NULL, NULL is returned. Monday is the first day of a week and the returned value is 0. Days are numbered in ascending order starting from 0. If the day is Sunday, the returned value is 6.

6.7.3.13. WEEKOFYEAR

Function declaration:

```
bigint weekofyear(datetime date)
```

Purpose: It is used to return the calendar week of the year that the specified date falls in. The system uses Monday as the first day of the week.

 **Note** If a week extends into the next year, the week belongs to the year containing four days or more. If more days fall in the first year, the week is considered as the last week of the first year. If more days fall in the second year, the week is considered as the first week of the second year.

Description:

date: the date of the datetime type. If the input is of the string type, it is implicitly converted to a value of the datetime type before this computation. For all other input types, an error is returned.

Returned value: bigint type. If the input is NULL, NULL is returned.

Example:

```
select weekofyear(to_date("20171229", "yyyymmdd")) from dual;
Returned value:
+-----+
| _c0    |
+-----+
| 1       |
+-----+
-- 20171229 is in year 2017, but the most days of the week are in year 2018. Therefore, the
returned value is 1, which indicates the first week of year 2018.
select weekofyear(to_date("20171231", "yyyymmdd")) from dual;
-- 1 is returned.
select weekofyear(to_date("20181229", "yyyymmdd")) from dual;
-- The returned value is 53.
```

6.7.3.14. Additional date functions

MaxCompute 2.0 provides additional date functions. You must add the following SET statement before SQL statements contained in the date functions:

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

 **Note** You must submit and execute the SET statement and the SQL statements of the new functions simultaneously.

Example:

```
set odps.sql.type.system.odps2=true;
select year('2017-01-01 12:30:00') = 2017 from dual;
```

The date functions described in subsequent topics are new in MaxCompute 2.0.

6.7.3.15. YEAR

Function declaration:

```
INT year(string date)
```

Purpose: It is used to return the year of the specified date.

Description:

date: the date of the string type. The date format must include yyyy-mm-dd and have no redundant strings. Otherwise, NULL is returned.

Returned value: int type.

Example:

```
year('2017-01-01 12:30:00') = 2017
year('2017-01-01') = 2017
year('17-01-01') = 17
year(2017-01-01) = null
year('2017/03/09') = null
year(null) = null
```

6.7.3.16. QUARTER

Command syntax:

```
int quarter(datetime/timestamp/string date)
```

Purpose: It is used to return the quarter of the input date, ranging from 1 to 4.

Description:

date: datetime, timestamp, or string type. The date format must include yyyy-mm-dd and have no redundant strings. Otherwise, NULL is returned.

Returned value: int type. If the input is NULL, NULL is returned.

Example:

```
quarter('2017-11-12 10:00:00') = 4
quarter('2017-11-12') = 4
```

6.7.3.17. MONTH

Function declaration:

```
INT month(string date)
```

Purpose: It is used to return the month of the input date.

Description:

date: This value must be a date of the string type. For all other input types, an error is returned.

Returned value: int type.

Example:

```
month('2017-09-01') = 9
month('20170901') = null
```

6.7.3.18. DAY

Function declaration:

```
INT day(string date)
```

Purpose: It is used to return the day of the input date.

Description:

date: This value must be a string type date. For all other input types, an error is returned.

Returned value: int type.

Example:

```
day('2017-09-01') = 1
day('20170901') = null
```

6.7.3.19. DAYOFMONTH

Function declaration:

```
INT dayofmonth(date)
```

Purpose: It is used to return the day of the month for the input date.

Description:

date: This value must be a string type date. For all other input types, an error is returned.

Returned value: int type.

Example:

```
dayofmonth('2017-09-01') = 1
dayofmonth('20170901') = null
```

6.7.3.20. HOUR

Function declaration:

```
INT hour(string date)
```

Purpose: It is used to return the hour of the input date.

Description:

date: This value must be a string type date. For all other input types, an error is returned.

Returned value: int type.

Example:

```
hour('2017-09-01 12:00:00') = 12
hour('12:00:00') = 12
hour('20170901120000') = null
```

6.7.3.21. MINUTE

Function declaration:

```
INT minute(string date)
```

Purpose: It is used to return the minute of the input date.

Description:

date: This value must be a string type date. For all other input types, an error is returned.

Returned value: int type.

Example:

```
minute('2017-09-01 12:30:00') = 30
minute('12:30:00') = 30
minute('20170901120000') = null
```

6.7.3.22. SECOND

Function declaration:

```
INT second(string date)
```

Purpose: It is used to return the second of the input date.

Description:

date: This value must be a string type date. For all other input types, an error is returned.

Returned value: int type.

Example:

```
second('2017-09-01 12:30:45') = 45
second('12:30:45') = 45
second('20170901123045') = null
```

6.7.3.23. FROM_UTC_TIMESTAMP

Function declaration:

```
timestamp from_utc_timestamp({any primitive type}*, string timezone)
```

Purpose: It is used to convert a UTC timestamp to a timestamp for a specified timezone.

Description:

- {any primitive type}*: the timestamp. The type can be `TIMESTAMP`, `DATETIME`, `TINYINT`, `SMALLINT`, `INT`, or `BIGINT`.
- `timezone`: Specifies the destination timezone, such as `PST`.

Returned value: `DATETIME` type.

Example:

```
select from_utc_timestamp(1501557840,'PST') = '1970-01-18 09:05:57.84'
select from_utc_timestamp('1970-01-30 16:00:00','PST') = '1970-01-30 08:00:00.0'
select from_utc_timestamp('1970-01-30','PST') = '1970-01-29 16:00:00.0'
```


6.7.3.24. CURRENT_TIMESTAMP

Function declaration:

```
timestamp current_timestamp()
```

Purpose: The current timestamp is returned as a Timestamp-type value. The value is not fixed.

Returned value: timestamp type.

Example:

```
select current_timestamp() from dual;  
-- '2017-08-03 11:50:30.661' is returned.
```

6.7.3.25. ADD_MONTHS

Function declaration:

```
string add_months(string startdate, int nummonths)
```

Purpose: It is used to return the date, which is 'nummonths' months later than 'startdate'.

Description:

- startdate: This value must be a string type date. The date format must contain yyyy-mm-dd. Otherwise, NULL is returned.
- num_months: int type.

Returned value: This value must be a string type date. The format is yyyy-mm-dd.

Example:

```
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
```

6.7.3.26. LAST_DAY

Function declaration:

```
string last_day(string date)
```

Purpose: It is used to return the last date of the month.

Description:

date: string type. The format is yyyy-MM-dd HH:mi:ss or yyyy-mm-dd.

Returned value: This value must be a datetime type date. The format is yyyy-mm-dd.

Example:

```
last_day('2017-03-04') = '2017-03-31'
last_day('2017-07-04 11:40:00') = '2017-07-31'
last_day('20170304') = null
```

6.7.3.27. NEXT_DAY

Function declaration:

```
string next_day(string startdate, string week)
```

Purpose: It is used to return the next date that is later than startdate and matches the week value. That is, the date of the day specified of the next week.

Description:

- startdate: string type. The format is yyyy-MM-dd HH:mi:ss or yyyy-mm-dd.
- week: string type. The name of a day, or the first 2 or 3 letters of the day, for example, Mo, TUE, or FRIDAY.

Returned value: This value must be a string type date. The format is yyyy-mm-dd.

Example:

```
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
```

6.7.3.28. MONTHS_BETWEEN

Function declaration:

```
double months_between(datetime/timestamp/string date1, datetime/timestamp/string date2)
```

Purpose: It is used to return the number of months between date1 and date2.

Description:

- date1: datetime, timestamp, or string type. The format is yyyy-MM-dd HH:mi:ss or yyyy-mm-dd.
- date2: datetime, timestamp, or string type. The format is yyyy-MM-dd HH:mi:ss or yyyy-mm-dd.

Returned value: double type.

- If 'date1' is later than 'date2', the returned value is positive. If 'date2' is later than 'date1', the returned value is negative.
- When date1 and date2 correspond to the last days of two months, the returned value is an integer representing the number of months. Otherwise, the formula is (date1 - date2)/31.

Example:

```
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
```

6.7.3.29. EXTRACT

Function declaration:

```
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 a time unit, such as YEAR, MONTH, DAY, HOUR, or MINUTE
- timestamp: a value of the TIMESTAMP type

Return value: A value of the INT type is returned.

Example:

```
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.

Example:

```
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 hour is 24, and the specified time value is 31. The return value is 7 (31/24).
-- The maximum value of minute is 60, and the specified time value is 61. The return value is 1 (61/60).
```

6.7.4. Window functions

6.7.4.1. Overview

In MaxCompute SQL statements, you can use the window function to analyze and process data flexibly. The window function can only appear in SELECT clauses. It does not support nested Window or aggregation functions. The window function cannot be used with the same-level aggregation functions at the same time.

A MaxCompute SQL statement supports up to five window functions.

Syntax:

```
window_func() over (partition by col1, [col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] windowing_clause)
```

Description:

- PARTITION BY specifies partition columns. The rows on which the partition column values are the same are considered to be in the same window. A window can contain up to 100 million rows of data (we recommend that the number of rows does not exceed 5 million). Otherwise, an error is returned.
- Use ORDER BY to specify the rule for sorting data in a window.
- You can use ROWS in windowing_clause to specify the partitioning method. There are two methods:
 - rows between x preceding|following and y preceding|following indicates a window range 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 range 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 10,000. 0 indicates the current row.
- You must specify ORDER BY before using ROWS to specify a window range.
- Not all window functions open windows using the method specified by ROWS. The method is only supported by the following functions: AVG, COUNT, MAX, MIN, STDDEV, and SUM.

6.7.4.2. COUNT

Command syntax:

```
bigint count([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

Purpose: It is used to return the number of values on the expr column.

Description:

- expr: any type. When it is NULL, this row is not involved in computation. If the distinct keyword is specified, this parameter indicates that only distinct values are counted.
- partition by col1[, col2...]: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: The count value of expr in the current window is returned if

ORDER BY is not set. The returned results are sorted in the specified order if ORDER BY is specified, and the value is the count value from the start row to the current row in the current window.

Returned value: bigint.

 **Note** If the distinct keyword is specified, ORDER BY cannot be used.

Example:

The user_id column of the bigint type exists in the test_src table.

```
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 number of values on the user_id column from the current partition is returned.
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 | -- start row of the window
| 1 | 2 | --two records exist from start row to current row. Return 2.
| 1       | 3     |
| 2       | 1     |
| 3       | 1     |
+-----+-----+
-- If ORDER BY is specified, the count value from the start row to the current row from the current partition is returned.
```

6.7.4.3. AVG

Function declaration:

```
avg([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc] [, col2[asc|desc]...]] [windowing_clause])
```

Purpose: It is used to calculate the average value.

Description:

- **distinct:** If the distinct keyword is specified, this parameter indicates that the average value of distinct values is calculated.
- **expr:** double type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before computation. If the input is of another type, an error is returned. If the input

is NULL, this row is not used in computation. The input cannot be of the boolean type.

- partition by col1[, col2]...: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: The count value of expr in the current window is returned if ORDER BY is not set. The returned results are sorted in the specified order if ORDER BY is specified, and the value is the count value from the start row to the current row in the current window.

Returned value: double type.

 **Note** If the distinct keyword is specified, ORDER BY cannot be set.

6.7.4.4. MAX

Function declaration:

```
max([distinct] expr) over(partition by col1[, col2...]  
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

Purpose: It is used to return the maximum value.

Description:

- expr: any types except the boolean type. If the value is NULL, the corresponding row is not involved in the operation. If the distinct keyword is specified, this parameter indicates that the maximum value of the distinct values is taken (whether this parameter is set or not does not affect the result).
- partition by col1[, col2...]: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: The maximum value in the current window is returned if ORDER BY is not set. If ORDER BY is set, the returned results are sorted in the specified order, and the values are the maximum values from the start row to the current row in the current window.

Returned value: The type is the same as that of expr.

 **Note** If the distinct keyword is specified, ORDER BY cannot be set.

6.7.4.5. MIN

Function declaration:

```
min([distinct] expr) over(partition by col1[, col2...]  
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

Purpose: It is used to return the minimum value.

Description:

- expr: any types except the boolean type. If the value is NULL, the corresponding row is not involved in the operation. If the distinct keyword is specified, this parameter indicates that the minimum value of distinct values is taken (whether this parameter is set or not does not affect the result).
- partition by col1[, col2...]: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: The minimum value in the current window is returned if ORDER BY is not set. If ORDER BY is set, the returned results are sorted in the specified order, and the returned value is the minimum value in the current window from the start row to the current row.

Returned value: The type is the same as that of expr.

 **Note** If the distinct keyword is specified, ORDER BY cannot be set.

6.7.4.6. MEDIAN

Function declaration:

```
double median(double number) over(partition by col1[, col2...])
decimal median(decimal number) over(partition by col1[,col2...])
```

Purpose: It is used to calculate the median.

Description:

- number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned. If the input is NULL, NULL is returned.
- partition by col1[, col2...]: specifies the partitions used in the computation.

Returned value: double type.

6.7.4.7. STDDEV

Function declaration:

```
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]...]] [windowi ng_clause])
```

Purpose: It is used to calculate the population standard deviation.

Description:

- expr: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other types of inputs, an error is returned. If the input value is NULL, then NULL is returned. If the distinct keyword is specified, this parameter indicates that the population standard deviation of distinct values is calculated.
- partition by col1[, col2...]: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: The population standard deviation of the current window is returned if ORDER BY is not set. If ORDER BY is set, the returned results are sorted in the specified order, and the values are the population standard deviation of the start row to the current row in the current window.

Returned value: When the input is of the decimal type, a value of the decimal type is returned. Otherwise, a value of the double type is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be set.

6.7.4.8. STDDEV_SAMP

Function declaration:

```
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])
```

Purpose: It is used to calculate the sample standard deviation.

Description:

- **expr:** double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other types of inputs, an error is returned. If the input is NULL, NULL is returned. If the distinct keyword is specified, this parameter indicates that the sample standard deviation of distinct values is calculated.
- **partition by col1[, col2...]:** specifies the partitions used in the computation.
- **order by col1 [asc|desc], col2[asc|desc]:** The sample standard deviation of the current window is returned if ORDER BY is not set. If ORDER BY is set, the returned results are sorted in the specified order, and the values are the sample standard deviation of the start row to the current row in the current window.

Returned value: When the input is of the decimal type, a value of the decimal type is returned. Otherwise, a value of the double type is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be set.

6.7.4.9. SUM

Function declaration:

```
sum([distinct] expr) over(partition by col1[, col2...]
[order by col1 [asc|desc] [, col2[asc|desc]...]] [windowing_clause])
```

Purpose: It is used calculate the sum.

Description:

- **expr:** double, decimal, or bigint type. If the input is of the string type, it is implicitly converted into a value of the double type before computation. If the input is of another type, an error is returned. If the value is NULL, this row is not calculated. If the distinct keyword is specified, this parameter indicates that the sum of distinct values is calculated.
- **partition by col1[, col2...]:** specifies the partitions used in the computation.
- **order by col1 [asc|desc], col2[asc|desc]:** The sum of the expr value in the current window is returned if ORDER BY is not set. If ORDER BY is set, the returned results are sorted in the order specified. The returned results are the cumulative sum of start row to the current row in the current window.

Returned value: When the input is of the bigint type, a value of the bigint type is returned. When the input is of the double or string type, a value of the double type is returned.

 **Note** If the distinct keyword is specified, ORDER BY cannot be set.

6.7.4.10. DENSE_RANK

Function declaration:

```
bigint dense_rank() over(partition by col1[, col2...]
order by col1 [asc|desc][, col2[asc|desc]...])
```

Purpose: It is used to calculate the consecutive ranking of values. The data in the same row of col2 has the same rank.

Description:

- partition by col1[, col2...]: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: This parameter specifies the value for deciding the ranking.

Returned value: bigint type.

Example:

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
```

To obtain their serial number, the employees must be group by their departments and sorted by SAL in descending order.

```

SELECT deptno
      , ename
      , sal
      , DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
-- DEPTNO (department) is the partition used in the computation, and SAL (salary) is used as
a basis for sorting returned results.
FROM emp;
-- Returned result:
+-----+-----+-----+-----+
| 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    |
+-----+-----+-----+-----+

```

6.7.4.11. RANK

Command syntax:

```
bigint rank() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

Purpose: It is used to return a ranking value. The ranking of the same row data with col2 drops.

Description:

- partition by col2[, col2..]: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: specifies the rule for deciding the ranking.

Returned value: bigint type.

Example:

Table emp 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

```

Now group the employees by department. Sort the employees in each group in descending order based on the salary. Each employee obtains a number that represents their position in the group.

```

SELECT deptno
      , ename
      , sal
      , RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
-- DEPTNO (department) is the partitioning column. The sal column is sorted to generate the
-- ranking value for each employee.
FROM emp;

```

-- Returned result:

```

+-----+-----+-----+-----+
| 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     |
+-----+-----+-----+-----+

```

6.7.4.12. LAG

Function declaration:

```
lag(expr, bigint offset, default) over(partition by col1[, col2...] [order by col1 [asc|desc] [, col2[asc|desc]...])
```

Purpose: It is used to retrieve the value in the row with a negative offset from the current row. For example, if the current row is m , the value retrieved is from the row $m - \text{offset}$.

Description:

- **expr:** any type.
- **offset:** a constant of the bigint type. If the input is of the string or double type, it is implicitly converted into a value of the bigint type before computation, and the offset is greater than 0.
- **default:** a constant. It specifies the default value when the offset is out of the valid range. The default value is NULL.
- **partition by col1[, col2...]:** specifies the partitions used in the computation.
- **order by col1 [asc|desc], col2[asc|desc]:** indicates the sorting order of the returned results.

Returned value: The type is the same as that of **expr**.

6.7.4.13. LEAD

Function declaration:

```
lead(expr, bigint offset, default) over(partition by col1[, col2...][order by col1 [asc|desc] [, col2[asc|desc]...])
```

Purpose: It is used to retrieve the value in the row with a positive offset from the current row. For example, if the current row is m , the value retrieved is from the row $m + \text{offset}$.

Description:

- **expr:** any type.
- **offset:** a constant of the bigint type. If the input is of the string or double type, it is implicitly converted into a value of the bigint type before computation, and the offset is greater than 0.
- **default:** a constant. It specifies the default value when the offset is out of the valid range.
- **partition by col1[, col2...]:** specifies the partitions used in the computation.
- **order by col1 [asc|desc], > col2[asc|desc]:** indicates the sorting order of the returned results.

Returned value: The type is the same as that of **expr**.

Example:

```
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 dual;
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 dual;
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 dual;
```

6.7.4.14. PERCENT_RANK

Function declaration:

```
percent_rank() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

Purpose: It is used to return the relative ranking of a row in a group of data.

Description:

- partition by col1[, col2...]: specifies the partitions used in the computation.
- order by col1 [asc|desc], col2[asc|desc]: specifies the value for the ranking.

Returned value: double type. Value range: 0 to 1. The relative ranking is calculated using the following formula: $(\text{rank}-1)/(\text{number of rows}-1)$.

 **Note** The number of rows in a window cannot exceed 10,000,000.

6.7.4.15. ROW_NUMBER

Function declaration:

```
row_number() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

Purpose: It is used to calculate the row number, which starts from 1.

Description:

- partition by col1[, col2...]: specifies the partitions used in the computation.
- order by col1 [asc|desc], > col2[asc|desc]: indicates the sorting value of the returned result.

Returned value: bigint type.

Example:

If table emp contains the following data:

```

| Empno | ename | job | Mgr | hiredate | Sal | REM | deptno |
7369, Smith, clerk, maid-12-17 00:00:00, 800, 20
7499, Allen, salesman, maid-02-20 00:00:00, 1600,300, 30
7521, Ward, salesman, maid-02-22 00:00:00, 1250,500, 30
7566, Jones, Manager, fig-04-02 00:00:00, 2975, 20
7654 Martin, salesman, fig-09-28 00:00:00, fig, 30
7698, Blake, Manager, fig-05-01 00:00:00, 2850, 30
7782, Clark, Manager, fig-06-09 00:00:00, 2450, 10
7788, Scott, analyst, fig-04-19 00:00:00, 3000, 20
00:00:00, King, President, 1991-11-17 5000, 7839, 10
7844, Turner, salesman, fig-09-08 00:00:00, 1500,0, 30
7876, Adams, clerk, maid-05-23 00:00:00, 1100, 20
7900 James, clerk, maid-12-03 00:00:00, 950, 30
7902 Ford, analyst, fig-12-03 00:00:00, 3000, 20
7934 Miller, clerk, fig-01-23 00:00:00, 1300, 10
7948, jaccka, clerk, fig-04-12 00:00:00, 5000, 10
7956, welan, clerk, fig-07-20 00:00:00, 2450, 10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10

```

Now, all employees need to be grouped by department, and each group must be sorted in descending order according to SAL to obtain the serial number in own group.

```

SELECT deptno
      , ename
      , sal
      , ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
-- DEPTNO (department) is the partition used in the computation, and SAL (salary) is used as
a basis for sorting results.
FROM emp;
-- Returned result:
+-----+-----+-----+-----+
| 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 |
+-----+-----+-----+-----+

```

6.7.4.16. CLUSTER_SAMPLE

Command syntax:

```
boolean cluster_sample(bigint x[, bigint y]) over(partition by col1[, col2..])
```

Purpose: It is used to conduct cluster sampling.

Description:

- **x:** bigint type. $x \geq 1$. If the parameter **y** is specified, **x** indicates that a window is divided into **x** parts. Otherwise, **x** indicates that **x** rows of records are extracted from a window (that is, the returned value is true if there are **x** rows). If **x** is NULL, NULL is returned.
- **y:** a constant of the bigint type. $y \geq 1$, $y \leq x$. This parameter extracts **y** records from **x** parts into which a window is divided (that is, the returned value is true if **y** records exist). If **y** is NULL, NULL is returned.
- **partition by col1[, col2]:** specifies the partitions used in the computation.

Returned value: boolean type.

Example:

The `test_tbl` table has two columns: `key` and `value`. The `key` column stores the group name of each value. The group names are `groupa` and `groupb`. The `value` column stores the values. The table structure is like this:

```
+-----+-----+
| 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 |
+-----+-----+
```

Run the following SQL statement to take a sample of 10% of the values in each group:

```
select key, value from (select key, value, cluster_sample(10, 1) over(partition by key) as
flag from tbl) sub where flag = true;
-- Returned result:
+-----+-----+
| key    | value                |
+-----+-----+
| groupa | -0.273930924365012  |
| groupb | -1.11828960785008  |
+-----+-----+
```

6.7.4.17. NTILE

Function declaration:

```
BIGINT ntile(BIGINT n) over(partition by coll[, col2...] [order by coll [asc|desc] [, col2[asc|desc]...]] [windowing_clause]))
```

Purpose: It is used to split grouped data into n slices and return the current slice number. If the slice is uneven, the distribution of the first slice is increased.

Description:

n: BIGINT type.

Returned value: BIGINT type.

Example:

Table emp has 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, sort each group in descending order by salary, and then obtain sequence numbers of employees in each group.


```
-- Execute the following statement:
select deptno, ename, sal, NTILE(3) OVER(PARTITION BY deptno ORDER BY sal desc) AS nt3 from
emp;
-- Returned result:
+-----+-----+-----+-----+
| 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        |
+-----+-----+-----+-----+
```

6.7.4.18. NTH_VALUE

Function declaration:

```
nth_value(expr, bigint n [, boolean skipNulls]) over(partition by coll[, col2...] order by co
ll [asc|desc][, col2[asc|desc]...])
```

Purpose: It is used to return the nth value in partitions used in the computation.

Description:

- **expr:** required. Any type.
- **n:** returns the nth value. It starts from 1 and is of the BIGINT type.
- **skipNulls:** specifies whether to ignore the rows whose values are NULL. This parameter is of the BOOLEAN type. The default value is false.

Returned value: the nth value in partitions used in the computation.

 **Note** If skipNulls is set to true, the nth non-NULL value is returned. If the nth non-NULL value does not exist, NULL is returned.

Example:

```

select a, nth_value(a + 1, 1) over (partition by a order by a) from values (3), (1), (2) as
t(a);
-- If n is 1, NTH_VALUE is equivalent to FIRST_VALUE.
-- Returned results:
-- 1      2
-- 2      3
-- 3      4

```

6.7.4.19. CUME_DIST

Function declaration:

```
cume_dist() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

Purpose: It is used to return the cumulative distribution. The cumulative distribution is the ratio between the number of rows whose values are less than or equal to the current value of the group and the total number of rows in the group.

Description: None.

 **Note** The order by column specifies values to be compared.

Returned value: the ratio of the number of rows whose values are equal to or less than the current value in the group to the total number of rows in the group.

Example:

Table emp has 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 then obtain the cumulative distribution of salary 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;
```

Returned result is as follows.

Returned result

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%

6.7.4.20. FIRST_VALUE

Function declaration:

```
first_value(expr) over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]
...])
```

Purpose: It is used to sort partitions and return the first value in the range from the beginning to the current row.

Description:

expr: required. Any type.

Returned value: the first expr value in partitions used in the computation.

Example:

Table emp has 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, sort each group in descending order by salary, and then obtain the name of the first employee in each group.

```
SELECT deptno
      , ename
      , sal
      , FIRST_VALUE(ename) OVER(PARTITION BY deptno ORDER BY sal desc) AS first1-- Obtain
the name of the first employee in each group after descending sorting by salary.
FROM emp;
```

Returned result is as follows.

Returned result

deptno	ename	sal	first1
10	JACCKA	5000.0	JACCKA
10	KING	5000.0	JACCKA
10	CLARK	2450.0	JACCKA
10	WELAN	2450.0	JACCKA

deptno	ename	sal	first1
10	TEBAGE	1300.0	JACCKA
10	MILLER	1300.0	JACCKA
20	SCOTT	3000.0	SCOTT
20	FORD	3000.0	SCOTT
20	JONES	2975.0	SCOTT
20	ADAMS	1100.0	SCOTT
20	SMITH	800.0	SCOTT
30	BLAKE	2850.0	BLAKE
30	ALLEN	1600.0	BLAKE
30	TURNER	1500.0	BLAKE
30	MARTIN	1250.0	BLAKE
30	WARD	1250.0	BLAKE
30	JAMES	950.0	BLAKE

6.7.4.21. LAST_VALUE

Function declaration:

```
last_value(expr) over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

Purpose: It is used to sort partitions and return the last value in the range from the beginning to the current row.

Description:

expr: required. Any type.

Returned value: the last expr value in partitions used in the computation.

Example:

Table emp has 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 then obtain the name of the last employee in each group.

```

SELECT deptno
      , ename
      , sal
      , LAST_VALUE(ename) OVER(PARTITION BY deptno ) AS last1
FROM emp;

```

The returned result is as follows.

Returned result

deptno	ename	sal	last1
10	TEBAGE	1300.0	WELAN
10	CLARK	2450.0	WELAN
10	KING	5000.0	WELAN
10	MILLER	1300.0	WELAN
10	JACCKA	5000.0	WELAN
10	WELAN	2450.0	WELAN
20	FORD	3000.0	JONES
20	SCOTT	3000.0	JONES
20	SMITH	800.0	JONES
20	ADAMS	1100.0	JONES

deptno	ename	sal	last1
20	JONES	2975.0	JONES
30	TURNER	1500.0	BLAKE
30	JAMES	950.0	BLAKE
30	ALLEN	1600.0	BLAKE
30	WARD	1250.0	BLAKE
30	MARTIN	1250.0	BLAKE
30	BLAKE	2850.0	BLAKE

6.7.5. Aggregate functions

6.7.5.1. Overview

An aggregate function aggregates multiple input records into an output record. The input is mapped many-to-one to the output. An aggregate function can be used with the GROUP BY clause at the same time.

6.7.5.2. COUNT

Command syntax:

```
bigint count([distinct|all] value)
```

Purpose: It is used to return the number of records.

Description:

- **distinct|all:** indicates whether duplicate records are cleared in counting. The default value is all, indicating that records are counted. If it is set to distinct, only records with distinct values are counted.
- **value:** any type. When it is NULL, this row is not involved in computation. value can be *. When it is set to count(*), the number of all rows is returned.

Returned value: bigint type.

Example:

In the tbla table, the col1 column is of the bigint type.

```

+-----+
| COL1 |
+-----+
| 1    |
+-----+
| 2    |
+-----+
| NULL |
+-----+
select count(*) from tbla;
-- 3 is returned.
select count(coll) from tbla;
-- The value is 2.

```

Aggregate functions can be used with the GROUP BY statement. For example, table test_src contains two columns: key (string type), and value (double type).

The data in the test_src table:

```

+-----+-----+
| key | value |
+-----+-----+
| a   | 2.0   |
+-----+-----+
| a   | 4.0   |
+-----+-----+
| b   | 1.0   |
+-----+-----+
| b   | 3.0   |
+-----+-----+
select key, count(value) as count from test_src group by key;
-- Run the preceding SQL statement. The output is:
+-----+-----+
| key | count |
+-----+-----+
| a   | 2     |
+-----+-----+
| b   | 2     |
+-----+-----+

```

Aggregate functions perform aggregation on values of the same key. The usage of the following aggregate functions is the same as that of this function and is not described in detail in this document.

6.7.5.3. AVG

Function declaration:

```

double avg(double value)
decimal avg(decimal value)

```

Purpose: It is used to calculate the average value.

Description:

value: double type or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned. If the value is NULL, this row is not used for calculation. The input cannot be of the boolean type.

Returned value: If the input is of the decimal type, a value of the decimal type is returned. For all other valid input types, a value of the double type is returned.

Example:

In the tbla table, the value column is of the bigint type.

```
+-----+
| value |
+-----+
| 1     |
| 2     |
| NULL  |
+-----+
select avg(value) as avg from tbla;
+-----+
| avg   |
+-----+
| 1.5   |
+-----+
-- The avg result of this column is as follows: (1 + 2) / 2 = 1.5.
```

6.7.5.4. MAX

Function declaration:

```
max(value)
```

Purpose: It is used to return the maximum value.

Description:

value: can be any data type. If the column value is NULL, the corresponding row is not involved in the operation. Values of the boolean type are excluded from the computation.

Returned value: The type is the same as that of value.

Example:

In the tbla table, the col1 column is of the bigint type.

```
+-----+
| col1 |
+-----+
| 1     |
| 2     |
| NULL  |
+-----+
select max(value) from tbla;
-- 2 is returned.
```

6.7.5.5. MIN

Function declaration:

```
MIN(value)
```

Purpose: It is used to return the minimum value.

Description:

value: a column of any data type. If a value in the column is NULL, the corresponding row is not involved in the operation. Boolean types are not allowed in this operation.

Returned value: The type is the same as that of value.

Example:

In the tbla table, the value column is of the bigint type.

```
+-----+
| value|
+-----+
| 1    |
+-----+
| 2    |
+-----+
| NULL |
+-----+
select min(value) from tbla;
-- 1 is returned.
```

6.7.5.6. MEDIAN

Function declaration:

```
double median(double number)
decimal median(decimal number)
```

Purpose: It is used to calculate the median.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other input types, an error is returned. If the input is NULL, a failure is returned.

Returned value: double or decimal type.

6.7.5.7. STDDEV

Function declaration:

```
double stddev(double number)
decimal stddev(decimal number)
```

Purpose: It is used to calculate the population standard deviation.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other types of inputs, an error is returned. If the input value is NULL, a failure is returned.

Returned value: double or decimal type.

6.7.5.8. STDDEV_SAMP

Function declaration:

```
double stddev_samp(double number)
decimal stddev_samp(decimal number)
```

Purpose: It is used to calculate the sample standard deviation.

Description:

number: double or decimal type. If the input is of the string or bigint type, it is implicitly converted into a value of the double type before this computation. For all other types of inputs, an error is returned. If the input is NULL, a failure is returned.

Returned value: double or decimal type.

6.7.5.9. SUM

Function declaration:

```
sum(value)
```

Purpose: It is used to calculate the sum.

Description:

value: double, decimal, or bigint type. If the input is of the string type, it is implicitly converted into a value of the double type before computation. If a value in the column is NULL, this row is not used for calculation. Values of the boolean type are excluded from calculation.

Returned value: When the input is of the bigint type, a value of the bigint type is returned. When the input is of the double or string type, a value of the double type is returned.

Example:

In the tbla table, the value column is of the bigint type.

```

+-----+
| value|
+-----+
| 1    |
+-----+
| 2    |
+-----+
| NULL |
+-----+
select sum(value) from tbla;
-- 3 is returned.

```

6.7.5.10. WM_CONCAT

Function declaration:

```
string wm_concat(string separator, string str)
```

Purpose: It is used to use the specified separator as the delimiter to link values in a string.

Description:

- **separator:** the delimiter, which is a constant of the string type. If it is of another type or is not a constant, an error is returned.
- **str:** string type. If the input is of the bigint, double, or datetime type, it is implicitly converted to a value of the string type before this computation. For all other input types, an error is returned.

Returned value: string type.

 **Note** If test_src in the select wm_concat(',', name) from > test_src; statement is an empty set, NULL is returned.

6.7.5.11. PERCENTILE

Function declaration:

```
DOUBLE percentile(BIGINT col, p)
array<double> percentile(BIGINT col, array(p1 [, p2]...))
```

Purpose: It is used to return the pth percentile of the specified column. p must be between 0 and 1.

 **Notice** You can only calculate true percentiles for integer values.

Description:

- **col:** BIGINT type.
- **p:** must be between 0 and 1.

Example:

Column c1 in table test has the following data:

```

+-----+
| c1      |
+-----+
| 8        |
| 9        |
| 10       |
| 11       |
+-----+

```

Calculate the pth percentile of column c1 in table test.

```

-- Execute the following statement:
select percentile(c1,0),percentile(c1,0.3),percentile(c1,0.5),percentile(c1,1) from test;
-- Returned result:
+-----+-----+-----+-----+
| _c0      | _c1      | _c2      | _c3      |
+-----+-----+-----+-----+
| 8.0       | 8.9       | 9.5       | 11.0      |
+-----+-----+-----+-----+

-- Execute the following statement:
select percentile(c1,array(0,0.3,0.5,1))from test;
-- Returned result:
+-----+
| _c0      |
+-----+
| [8, 8.9, 9.5, 11] |
+-----+

```

6.7.5.12. Additional aggregate functions

MaxCompute 2.0 provides additional aggregate functions. You must add the following SET statement before SQL statements contained in the aggregate functions:

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

 **Note** You must submit and execute the SET statement and the SQL statements of the new functions simultaneously.

The aggregate functions described in subsequent topics are new in MaxCompute 2.0.

6.7.5.13. COLLECT_LIST

Command syntax:

```
ARRAY collect_list(col)
```

Purpose: It is used to convert the values on the col column into an array.

Description:

col: a table column of any data type.

Returned value: array type.

6.7.5.14. COLLECT_SET

Command syntax:

```
ARRAY collect_set(col)
```

Purpose: It is used to convert the values on the col column with duplicates removed into an array.

Description:

col: a table column of any data type.

Returned value: array type.

6.7.5.15. VARIANCE/VAR_POP

Function declaration:

```
DOUBLE variance(col)
DOUBLE var_pop(col)
```

Purpose: It is used to calculate the variance of the specified numeric column.

Description:

col: numeric type column. NULL is returned for other types.

Returned value: DOUBLE type.

Example:

Column c1 in table test has the following data:

```
+-----+
| c1    |
+-----+
| 8     |
| 9     |
| 10    |
| 11    |
+-----+
```

Calculate the variance of column c1 in table test.

```
-- Execute the following statement:
select variance(c1) from test;
-- or
select var_pop(c1) from test;
-- Returned result:
+-----+
| _c0    |
+-----+
| 1.25    |
+-----+
```

6.7.5.16. VAR_SAMP

Function declaration:

```
DOUBLE var_samp(col)
```

Purpose: It is used to calculate the sample variance of the specified numeric column.

Description:

col: numeric type column. NULL is returned for other types.

Returned value: DOUBLE type.

Example:

Column c1 in table test has the following data:

```
+-----+
| c1    |
+-----+
| 8     |
| 9     |
| 10    |
| 11    |
+-----+
```

Calculate the variance of column c1 in table test.

```
-- Execute the following statement:
select var_samp(c1) from test;
-- Returned result:
+-----+
| _c0    |
+-----+
| 1.666666666666667 |
+-----+
```

6.7.5.17. COVAR_POP

Function declaration:

```
DOUBLE covar_pop(col1, col2)
```

Purpose: It is used to calculate the population covariance of two specified numeric columns.

Description:

col1 and col2: numeric type columns. NULL is returned for other types.

Example:

Columns c1 and c2 in table test have the following data:

```

+-----+-----+
| c1      | c2      |
+-----+-----+
| 3        | 2        |
| 14       | 5        |
| 50       | 14       |
| 26       | 75       |
+-----+-----+

```

Calculate the population covariance of columns c1 and c2.

```

-- Execute the following statement:
select covar_pop(c1,c2) from test;
-- Returned result:
+-----+
| _c0      |
+-----+
| 123.49999999999997 |
+-----+

```

6.7.5.18. COVAR_SAMP

Function declaration:

```
DOUBLE covar_samp(col1, col2)
```

Purpose: It is used to calculate the sample covariance of two specified numeric columns.

Description:

col1 and col2: numeric type columns. NULL is returned for other types.

Example:

Columns c1 and c2 in table test have the following data:

```

+-----+-----+
| c1      | c2      |
+-----+-----+
| 3        | 2        |
| 14       | 5        |
| 50       | 14       |
| 26       | 75       |
+-----+-----+

```

Calculate the sample covariance of columns c1 and c2.


```
-- Execute the following statement:
select covar_samp(c1,c2) from test;
-- Returned result:
+-----+
| _c0      |
+-----+
| 164.66666666666663|
+-----+
```

6.7.6. Other functions

6.7.6.1. ARRAY

Function declaration:

```
array(value1,value2, ...)
```

Purpose: It is used to create an array by using input values.

Description:

value: any type. All the values must be of the same type.

Returned value: ARRAY type.

Example:

```
select array(123,456,789) from dual;
-- Returned result:
[123, 456, 789]
```

6.7.6.2. ARRAY_CONTAINS

Function declaration:

```
array_contains(ARRAY<T> a, value v)
```

Purpose: It is used to check whether array a contains value v.

Description:

- a: array type.
- v: The given value v must be of the same type as the data in the array.

Returned value: boolean type.

Example:

```
select array_contains(array('a','b'), 'a') from dual;
-- True is returned.
select array_contains(array(456,789),123) from dual;
-- False is returned.
```

6.7.6.3. CAST

Command syntax:

```
cast(expr as <type>)
```

Purpose: It is used to convert an expression of one data type to another. For example, cast ('1' as bigint) converts 1 of the string type to the integer type. If the conversion fails, an error is returned.

 Note

- cast(double as bigint) converts a value of the double type into a value of the bigint type.
- cast(string as bigint) converts a value of the string type into a value of the bigint type. If the string is composed of numerals expressed in integer form, it is directly converted into a value of the bigint type. If the string is comprised of numerals expressed in the 'float' or 'exponent' form, it is converted to 'double' type first and then to 'bigint' type.
- For cast(string as datetime) or cast(datetime as > string), the datetime format is yyyy-mm-dd hh:mi:ss by default.

6.7.6.4. COALESCE

Command syntax:

```
coalesce(expr1, expr2, ...)
```

Purpose: It is used to return the first non-NULL value in the list. If all values in the list are NULL, NULL is returned.

Description:

expr: a value to be tested. All these values must be of the same type or be NULL. Otherwise, an error is returned.

Returned value: The type is the same as that of the input.

 **Note** At least one parameter is provided. Otherwise, an error is returned.

6.7.6.5. DECODE

Function declaration:

```
decode(expression, search, result[, search, result]...[, default])
```

Purpose: It is used to implement the if-then-else conditional branching feature.

Description:

- expression: expression to be compared.
- search: search string to be compared with the expression.
- result: the value returned when the value of search matches the expression.
- default: optional. If no search string matches the expression, the default value is returned. If it is not

specified, NULL is returned.

Returned value: The matched search is returned. If there are no matches, the default value is returned. If default is not specified, NULL is returned.

Note

- At least three parameters are specified.
- All results must share the same type or be NULL. Inconsistent data types will cause an error. All values of search and expression must be of the same type. Otherwise, an error is returned.
- If the search option in decode has repeated records and matches the expression, the first search value is returned.

Example:

```
select decode(customer_id,
1, 'Taobao',
2, 'Alipay',
3, 'Aliyun',
NULL, 'N/A',
'Others') as result from sale_detail;
```

The preceding DECODE function implements the feature in the following 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;
```



Notice The MaxCompute SQL statement returns NULL when calculating NULL = NULL. However, in the DECODE function, values of NULL and NULL are equal. In the preceding example, when the value of customer_id is NULL, the DECODE function returns N/A.

6.7.6.6. EXPLODE

Function declaration:

```
explode (var)
```

Purpose: It is used to convert one row of data into multiple rows of UDTF. If var is of the array type, the array stored in the column is converted into multiple rows. If var is of the map type, each key-value pair of the map stored in the column is converted into a row with two columns, with one column for the key and the other for the value.

Description:

var: array < T > type or map < K,V > type.

Returned value: transposed rows.

Note

Limits on the use of UDTFs:

- Only one UDTF is allowed in a SELECT statement, and other columns are not allowed.
- One select can only have one UDTF and no other columns can appear.

Example:

```
explode(array(null, 'a', 'b', 'c')) col
```

6.7.6.7. GET_IDCARD_AGE

Function declaration:

```
get_idcard_age(idcardno)
```

Purpose: It is used to return the current age based on the ID card number. The current age is the current year minus the birth year on the ID card.

Description:

idcardno: string type, ID number of 15-digit or 18-digit. During the calculation, 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.

Returned value: bigint type. If the input is NULL, NULL is returned. If the difference of the current year minus the birth year is greater than 100, then NULL is returned.

6.7.6.8. GET_IDCARD_BIRTHDAY

Function declaration:

```
get_idcard_birthday(idcardno)
```

Purpose: It is used to return the date of birth based on the ID card number.

Description:

idcardno: string type, a 15-digit or 18-digit ID card number. During 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.

Returned value: datetime type. If the input is NULL, NULL is returned.

6.7.6.9. GET_IDCARD_SEX

Function declaration:

```
get_idcard_sex(idcardno)
```

Purpose: It is used to return the gender based on the ID card number. The returned value is M (male) or F (female).

Description:

idcardno: string type, a 15-digit or 18-digit ID card number. During 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.

Returned value: string type. If the input is NULL, NULL is returned.

6.7.6.10. GREATEST

Function declaration:

```
greatest(var1, var2, ...)
```

Purpose: It is used to return the maximum value among the input values.

Description:

var: bigint, double, datetime, or string type. If all values are NULL, NULL is returned.

Returned value:

- The greatest value in input parameter. If the implicit conversion is not needed, return type is the same as input parameter type.
- NULL is interpreted as the minimum value.
- If the input parameters are of different types, values of the double, bigint, and string types are converted into values of the double type for comparison, and values of the string and datetime types are converted into values of the datetime type for comparison. Implicit conversion of other types is not allowed.

6.7.6.11. INDEX

Function declaration:

```
index(var1[var2])
```

Purpose: It is used to return the specified element in a given array, or return the value of the specified key in a given map.

Description:

- var1: array < T > type or map < K,V > type.
- var2: If var1 is of the array < T > type, var2 must be the bigint type must be larger or equal to 0. If var1 is of the map < K,V > type, var2 is of the K type.

Returned value:

- If var1 is of the array < T > type, a value of the T type is returned. If var2 is out of range of array < T > elements, NULL is returned.
- If var1 is of the map < K,V > type, a value of the V type is returned. If no key is var2 in map < K,V >, NULL is returned.

Example:

If var1 is an array, run the following SQL statement:

```
select array('a','b','c')[2] from dual;
-- Returned result:
+-----+
| _c0 |
+-----+
| c   |
+-----+
```

If var1 is of the map type, run the following SQL statement:

```
select str_to_map("test1=1,test2=2")["test1"] from dual;
-- Returned result:
+-----+
| _c0 |
+-----+
| 1   |
+-----+
```



Notice

- To use the SQL statement, remove the index and run var1[var2] directly. Otherwise, a syntax error is returned.
- If Var1 is NULL, NULL is returned.

6.7.6.12. MAX_PT

Function declaration:

```
max_pt(table_full_name)
```

Purpose: For partitioned tables, it is used to return the maximum values in the first-level partitions that have data files and sort the values in alphabetic order.

Description:

table_full_name: string type. It specifies a table name (project name required, for example, prj.src). You must have the read permission on the table.

Returned value: maximum value in the primary partition.

Example:

Partitioned table tbl has the following partitions with data files: pt='20170901' and pt='20170902'. In the following statement, the returned value of max_pt is '20170902'. The MaxCompute SQL statement reads data from the '20170902' partition.

```
select * from tbl where pt=max_pt('myproject.tbl');
```



Note If a new partition is added by using alter table, but there is no data file in this partition, then this partition is not returned.

6.7.6.13. ORDINAL

Function declaration:

```
ordinal(bigint nth, var1, var2, ...)
```

Purpose: It is used to sort the input variables in ascending order, and return the specified nth value.

Description:

- nth: bigint type. It specifies the position at which the value is to be returned. If it is NULL, NULL is returned.
- var: bigint, double, datetime, or string type.

Returned value:

- The value in nth bit. If the implicit conversion is not needed, return type is the same as input parameter type.
- If type conversion is performed, values of the double, bigint, and string types are converted into values of the double type. Values of the string and datetime types are converted into values of the datetime type. Implicit conversion of other types is not allowed.
- NULL is the least value.

Example:

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) = 2
```

6.7.6.14. LEAST

Function declaration:

```
least(var1, var2, ...)
```

Purpose: It is used to returns the minimum value among the input values.

Description:

var: bigint, double, datetime, or string type. If all values are NULL, NULL is returned.

Returned value:

- The least value in input parameter; If the implicit conversion is not needed, return type is the same as input parameter type.
- If type conversion is performed, values of the double, bigint, and string types are converted into values of the double type. Values of the string and datetime types are converted into values of the datetime type. Implicit conversion of other types is not allowed.
- NULL is interpreted as the minimum value.

6.7.6.15. SIZE

Function declaration:

```
size(map<K, V>)  
size(array<T>)
```

Purpose: size(map) is used to return the number of key-value pairs in the given map, and size(array) is used to return the number of elements in the given array.

Description:

- map: map type.
- array: array type.

Returned value: int type.

Example:

```
select size(map('a',123,'b',456)) from dual;
-- 2 is returned.
select size(map('a',123,'b',456,'c',789)) from dual;
-- 3 is returned.
select size(array('a','b')) from dual;
-- 2 is returned.
select size(array(123,456,789)) from dual;
-- 3 is returned.
```

6.7.6.16. SPLIT

Function declaration:

```
split(str, pat)
```

Purpose: It is used to split a string using the specified separator.

Description:

- str: string type. The string to be separated.
- pat: string type. It indicates the separator and supports regular expressions.

Returned value: array <string>. The returned array contains elements extracted from the string based on the specified separator.

Example:

```
select split("a,b,c",",") from dual;
-- Returned result:
+-----+
| _c0 |
+-----+
| [a, b, c] |
+-----+
```

6.7.6.17. STR_TO_MAP

Function declaration:

```
str_to_map(text [, delimiter1 [, delimiter2]])
```


Purpose: It is used to divide 'text' into K-V pairs with 'delimiter1', and to separate each K-V pair with 'delimiter2'.

Description:

ext: string type. It indicates the string to be separated.

delimiter1: string type. It is the delimiter. If it is not specified, the default value ',' is used.

delimiter2: string type. It is the delimiter. If it is not specified, the default value '=' is used.

Returned value: map < string, string >. The elements are the K-V results of the separation of 'text' by the strings 'delimiter1' and 'delimiter2'.

Example:

```
select str_to_map("test1=1,test2=2") from dual;
-- Returned result:
+-----+
| a      |
+-----+
| {Test1: 1, Test2: 2} |
```

6.7.6.18. UNIQUE_ID

Function declaration:

```
STRING UNIQUE_ID()
```

Purpose: It is used to return a random but unique ID, for example, 29347a88-1e57-41ae-bb68-a9edbdd94212_1. This function runs more efficiently than UUID.

6.7.6.19. UUID

Function declaration:

```
string uuid()
```

Purpose: It returns a random ID, for example, 29347a88-1e57-41ae-bb68-a9edbdd94212.

6.7.6.20. SAMPLE

Function declaration:

```
boolean sample(x, y, column_name)
```

Purpose: It is used to sample all values read from the specified column based on the given settings, and filters out the rows that do not meet the sampling condition.

Description:

- x, y: bigint type. It indicates that data is hashed to x portions and the yth portion is taken. y can be omitted. If y is omitted, the first portion is taken and column_name must also be omitted. x and y are constants of the integer type and are greater than 0. If they are of another type or if they are less than or equal to 0, an error is returned. If y is greater than x, an error is returned. If either x or y is NULL,

NULL is returned.

- **column_name**: target column of sampling. **column_name** can be omitted. If **column_name** is omitted, random sampling is performed based on values of **x** and **y**. It can be of any type, and the column value can be NULL. No implicit conversion is performed. If **column_name** is the constant NULL, an error is reported.

Returned value: boolean type.

 **Note** To avoid data skew resulting from the NULL value, a uniform hash of **x** is made for a value of NULL in **column_name**. If **column_name** is not added, the output is not necessarily uniform since the data size is smaller. So **column_name** is suggested to be added to get better output.

Example:

Table **tbla** contains a column named **cola**.

```
select * from tbla where sample (4, 1 , cola) = true;
-- The values are hashed to four portions based on cola, and the first portion is used.
select * from tbla where sample (4, 2) = true;
-- The values in each row are randomly hashed to four portions, and the second portion is used.
```

6.7.6.21. CASE WHEN expression

MaxCompute provides the following two kinds of CASE WHEN syntax formats:

```
case value
when (_condition1) then result1
when (_condition2) then result2
...
else resultn
end
case
when (_condition1) then result1
when (_condition2) then result2
when (_condition3) then result3
...
else resultn
end
```

CASE WHEN flexibly returns different values based on the calculation result of the expression. Alibaba Cloud StreamCompute supports two types of CASE WHEN expressions:

```
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 there are values of only the bigint and double type in the results, the results are converted into values of the double type.
- If there is a value of the string type in the results, the results are all converted into values of the string type. If the result of a type cannot be converted (for example, boolean type), an error is returned.
- Conversion between other types is not allowed.

6.7.6.22. IF

Function declaration:

```
if(testCondition, valueTrue, valueFalseOrNull)
```

Purpose: It is used to determine whether 'testCondition' is true. If it is true, valueTrue is returned. If it is not true, valueFalseOrNull is returned.

Description:

testCondition: boolean type. The expression to be determined true or not.

valueTrue: the value returned when expression 'testCondition' is true.

valueFalseOrNull: the value returned when expression 'testCondition' is false. It can be set to NULL.

Returned value: The type is the same as that of valueTrue or valueFalseOrNull.

Example:

```
select if(1=2,100,200) from dual;
-- Returned result:
+-----+
| _c0    |
+-----+
| 200    |
+-----+
```

6.7.6.23. Additional functions

MaxCompute 2.0 provides additional functions.

The functions described in the following topics are new in this version.

6.7.6.24. MAP

Function declaration:

```
map(K key1, V value1, K key2, V value2, ...)
```

Purpose: It is used to create a map with the given K-V pairs.

Description:

key/value: The types of all keys are the same and must be of one of the basic types. The types of all values are the same and can be of any type.

Returned value: map type.

Example:

```
select map('a',123,'b',456) from dual;
-- Returned result:
{a:123, b:456}
```

6.7.6.25. MAP_KEYS

Function declaration:

```
map_keys (map<K, V> )
```

Purpose: It is used to return all keys in the map parameter as an array.

Description:

map: data of the map type.

Returned value: array type. If the input is NULL, NULL is returned.

Example:

```
select map_keys(map('a',123,'b',456)) from dual;
-- Returned result:
[a, b]
```

6.7.6.26. MAP_VALUES

Function declaration:

```
map_values (map<K, V>)
```

Purpose: It is used to return all values in the map parameter as an array.

Description:

map: map type.

Returned value: array type. If the input is NULL, NULL is returned.

Example:

```
select map_values(map('a',123,'b',456)) from dual;
-- Returned result:
[123, 456]
```

6.7.6.27. SORT_ARRAY

Function declaration:

```
sort_array(ARRAY<T>)
```

Purpose: It is used to sort a given array.

Description:

ARRAY: array type. The data in the array is of any type.

Returned value: array type.

Example:

```
select sort_array(array('a','c','f','b')),sort_array(array(4,5,7,2,5,8)),sort_array(array('
You','Me','He')) from dual;
-- Returned result:
[a, b, c, f] [2, 4, 5, 5, 7, 8] [him, you, me]
```

6.7.6.28. POSEXplode

Command syntax:

```
posexplode(ARRAY<T>)
```

Purpose: It is used to explode the given array. Each value is given a row and each row has two columns corresponding to the subscript (starting from 0) and the array element.

Description:

ARRAY: array type. Data in the array can be of any type.

Returned value: table generation function.

Example:

```
select posexplode(array('a','c','f','b')) from dual;
-- Returned result:
+-----+-----+
| pos    | val  |
+-----+-----+
| 0      | a    |
| 1      | c    |
| 2      | f    |
| 3      | b    |
+-----+-----+
```

6.7.6.29. STRUCT

Function declaration:

```
struct(value1,value2, ...)
```

Purpose: It is used to create a struct using a given value list.

Description:

value: any type.

Returned value: struct type. The field names of the created struct are col1, col2, and so on.

Example:

```
select struct('a',123,'ture',56.90) from dual;
-- Returned result:
{col1:a, col2:123, col3:true, col4:56.9}
```

6.7.6.30. NAMED_STRUCT

Function declaration:

```
named_struct(string name1, T1 value1, string name2, T2 value2, ...)
```

Purpose: It is used to create a struct using a given name-value list.

Description:

- value: any type.
- name: field name of the string type.

Returned value: struct type. The field names of the created struct are name1, name2, and so on.

Example:

```
select named_struct('user_id',10001,'user_name','bob','married','F','weight',63.50) from dual;
-- Returned result:
{user_id:10001, user_name:bob, married:F, weight:63.5}
```

6.7.6.31. INLINE

Function declaration:

```
inline(ARRAY<STRUCT<f1:T1, f2:T2, ... >>)
```

Purpose: It is used to expand a struct, with each element corresponding to a row, and each struct element in each row corresponding to a column.

Description:

STRUCT: The values in the array can be of any type.

Returned value: table generation function.

Example:

```
select inline(array(named_struct('user_id',10001,'user_name','bob','married','F','weight',63.50))) from dual;
-- Returned result:
+-----+-----+-----+-----+
| user_id | user_name | married | weight |
+-----+-----+-----+-----+
| 10001   | bob      | F      | 63.5   |
+-----+-----+-----+-----+
```

6.7.6.32. BETWEEN AND expression

Command syntax:

```
A [NOT] BETWEEN B AND C
```

If A, B, or C is NULL, then the value is NULL. If A is greater than or equal to B, and less than or equal to C, the value is true. Otherwise, the value is false.

Example:

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
```

Run the following command to query data where sal is greater than or equal to 1,000 and less than or equal to 1,500:

```
select * from emp where sal BETWEEN 1000 and 1500;
-- Returned result:
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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 |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

6.7.6.33. NVL

Function declaration:

```
nvl(T value, T default_value)
```

Purpose: It is used to return default_value if value is NULL and return value otherwise.

Example:

Table t_data has three columns of c1 string, c2 bigint, and c3 datetime, as well as the following data:

```
+-----+-----+-----+
| 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 |
+-----+-----+-----+
```

Use the NVL function to output the NULL values in c1 to 00000, the NULL values in c2 to 0, and the NULL values in c3 to "-".

```
-- 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 |
+-----+-----+-----+
```

6.7.6.34. TABLE_EXISTS

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:

```
-- Used in a SELECT statement.
SELECT IF(table_exists('abd'), col1, col2) FROM src;
-- Used in an IF-ELSE branch statement.
IF (table_exists('abd'))
-- statements
ELSE
-- statements
```

6.7.6.35. PARTITION_EXISTS

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. Set this parameter to the partitioning column values based on partition key columns in sequence. The number of partition names must be the same as that 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 ds='20190101' and hr='1' exist.
```

6.8. UDFs

6.8.1. Overview

UDF is short for user defined function. MaxCompute provides a variety of built-in functions. You can also create UDFs based on specific computing requirements. You can use UDFs as using common built-in functions. This topic briefs how to use SQL UDFs. For more information about SQL UDFs, see the official documentation on UDFs.

The following table lists the extended UDFs in MaxCompute.

UDF category

UDF category	Description
UDF	User defined scalar functions are commonly referred to as UDFs. There is a one-to-one mapping between the input and output. Each time a UDF reads a row of data, it writes an output value.
UDTF	User defined table valued functions are commonly referred to as UDTFs. Each time a UDTF is called, it outputs multiple rows of data. UDTFs are the only category that returns multiple fields. A UDF only returns one value each time.
UDAF	User defined aggregation functions are commonly referred to as UDAFs. A UDAF aggregates multiple input records into one output record. There is a many-to-one mapping between input and output. A UDAF can be used together with the GROUP BY clause (SQL) at the same time. For more information about the syntax, see aggregation functions.

 **Note** In general, UDFs refer to all user defined functions: UDFs, UDAFs, and UDTFs. In a narrow sense, UDFs only refer to user defined scalar functions. This term is used interchangeably in this document. You will have to determine the exact meaning based on the context.

6.8.2. Types of parameters and returned values

UDFs support the following MaxCompute SQL data types:

- Basic data types: BIGINT, DOUBLE, BOOLEAN, DATETIME, DECIMAL, STRING, TINYINT, SMALLINT, INT, FLOAT, VARCHAR, BINARY, and TIMESTAMP.
- Complex data types: ARRAY, MAP, and STRUCT.

 **Note** In UDFs, you can define the writable attribute of parameters.

The usage of some basic data types (such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, BINARY, and TIMESTAMP) in Java UDFs is as follows:

- UDAFs and UDTFs use the @Resolve annotation to obtain signatures. Example: `@Resolve("smallint ->varchar(10)")`.
- UDFs reflect and analyze the evaluate() method to obtain signatures. In this case, there is a one-to-one mapping between MaxCompute built-in types and Java types.

To use complex data types (ARRAY, MAP, and STRUCT) in Java UDFs, take the following steps:

- UDTFs use the `@Resolve` annotation to specify signatures. Example: `@Resolve("array<string>, struct<al:bigint,b1:string>, string->map<string, bigint>, struct<b1:bigint>")`.
- UDFs use the signature of the `evaluate()` method to map the input and output types. For more information, see the mappings between MaxCompute types and Java types. In the preceding example, `ARRAY` corresponds to `java.util.List`, `MAP` corresponds to `java.util.Map`, and `STRUCT` corresponds to `com.aliyun.odps.data.Struct`.
- UDAFs and UDTFs use the `@Resolve` annotation to obtain signatures. Example: `@Resolve("smallint->varchar(10)")`.



Notice

- You can use `type, *` to add any number of parameters. Example: `@resolve("string,*->array<string>")`. Note that you must add a subtype after array.
- The field name and field type of `com.aliyun.odps.data.Struct` cannot be reflected. Therefore, the `@Resolve` annotation is required. If you want to use `struct` in a UDF, you must add the `@Resolve` annotation to the UDF class. This annotation only affects the overloads of parameters or returned values that contain `com.aliyun.odps.data.Struct`.
- A class supports only one `@Resolve` annotation. A UDF that contains `struct` can only reload parameters or returned values once.

The following table lists the mapping between MaxCompute and Java data types.

Data type mapping

MaxCompute type	Java type
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL	java.math.BigDecimal
BOOLEAN	java.lang.Boolean
STRING	java.lang.String
VARCHAR	com.aliyun.odps.data.Varchar
BINARY	com.aliyun.odps.data.Binary
DATETIME	java.util.Date
TIMESTAMP	java.sql.Timestamp

MaxCompute type	Java type
ARRAY	java.util.List
MAP	java.util.Map
STRUCT	com.aliyun.odps.data.Struct

Note

- Java data types and the data types of returned values are objects, and must start with a capitalized letter.
- The NULL value in SQL is represented by a NULL reference in Java. The Java primitive type is not allowed because it cannot represent a NULL value in SQL.
- The ARRAY type in MaxCompute corresponds to a list, not an array, in Java.

The following table compares the API features of two languages.

API feature comparison

Supported language	UDF	UDAF	UDTF	DATETIME type	Read resource file	Read resource table
Python	Yes	Yes	Yes	Yes	Yes	Yes
Java	Yes	Yes	Yes	Yes	Yes	Yes

6.8.3. UDFs

A UDF must inherit the `com.aliyun.odps.udf.UDF` class and implement the `EVALUATE` method. The `EVALUATE` method must be a non-static public method. The types of parameters and returned values of the `EVALUATE` method are used as the UDF signatures in SQL. This means that users can implement multiple `EVALUATE` methods in a UDF. When a UDF is called, the framework matches the correct `EVALUATE` method based on the parameter type called by the UDF.

Example:

```
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();
}
}
```

 **Note** You can implement `void setup(ExecutionContext ctx)` and `void close()` to implement UDF initialization and termination code, respectively.

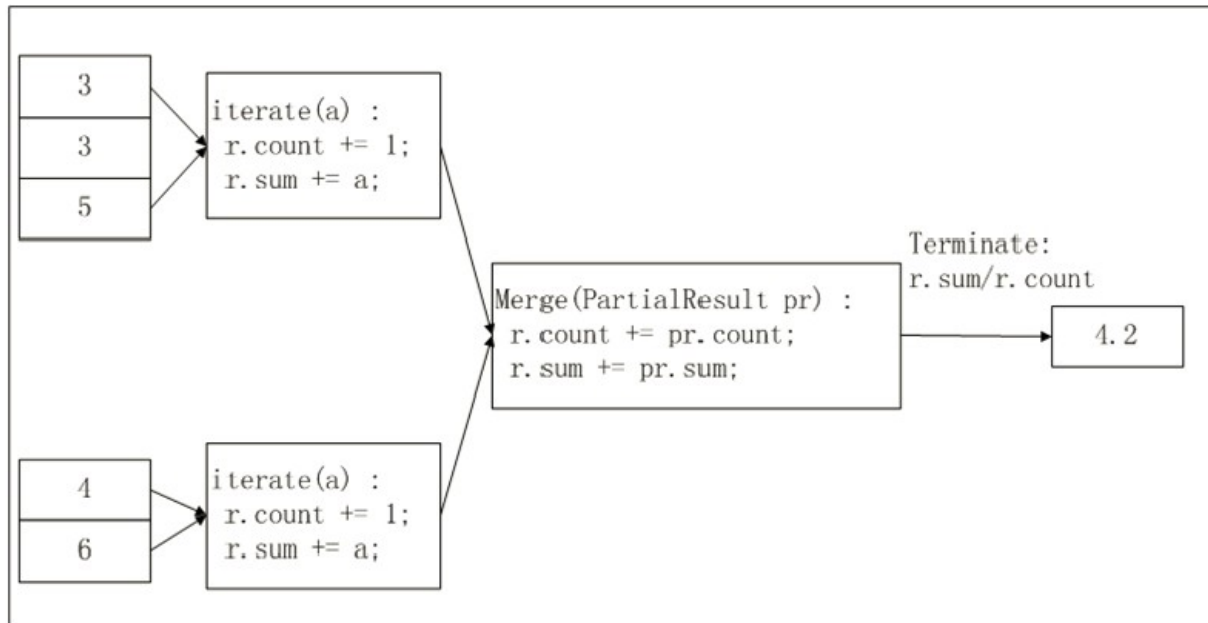
UDFs are used in the same way as built-in functions in MaxCompute SQL. For more information, see [Built-in functions](#).

6.8.4. UDAFs

To implement a Java UDAF, you must inherit the `com.aliyun.odps.udf.UDAF` class and implement the following APIs:

```
public abstract class Aggregator implements ContextFunction {
    @Override
    public void setup(ExecutionContext ctx) throws UDFException {
    }
    @Override
    public void close() throws UDFException {
    }
    /**
     * Create an aggregate buffer
     * @return Writable - Aggregate buffer
     */
    abstract public Writable newBuffer();
    /**
     * @param buffer - Aggregate buffer
     * @param args - Parameter specified when SQL calls UDAFs
     * @throws UDFException
     */
    abstract public void iterate(Writable buffer, Writable[] args) throws UDFException;
    /**
     * generate final result
     * @param buffer
     * @return final result of Object UDAF
     * @throws UDFException
     */
    abstract public Writable terminate(Writable buffer) throws UDFException;
    abstract public void merge(Writable buffer, Writable partial) throws UDFException;
}
```

The most important APIs are `iterate`, `merge`, and `terminate`. The primary logic of UDAFs relies on the implementation of these three APIs. In addition, you must implement a custom writable buffer. As an example, the following figure briefly illustrates the implementation logic and computational flow of the `avg` (average value) MaxCompute UDAF function.



In the preceding figure, the input data is sliced by a certain size (for description of slicing, see [MapReduce](#)). The size of each slice is suitable for a worker to complete in an appropriate period of time. You need to manually configure the size of the slices.

The UDAF calculation process is divided into two phases:

- Phase 1: Each Worker counts the number of data rows and the sum of the data in each slice. The user can regard the counted number and sum as an intermediate result.
- Phase 2: The Worker summarizes the information gained from the previous phase within each slice. In the final output, $r.sum / r.count$ is the average of all input data.

The following example shows how to calculate an average by using a UDAF:

```

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;
    }
    }

```

Notice

- The SQL syntax used by UDAFs is the same as that used by common built-in aggregate functions. For

more information, see [Aggregate functions](#).

- The way to run UDTFs is the same as that to run UDFs. For more information, see [Run UDFs](#).

6.8.5. UDTFs

6.8.5.1. Overview

Java UDTFs must inherit the `com.aliyun.odps.udf.UDTF` class. This class requires the implementation of four APIs. The following table lists the definitions of these APIs.

API definitions

API	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. SETUP is called once first in each worker.
<code>public void process(Object[] args) throws UDFException</code>	This method is called by the framework. Each SQL record calls PROCESS once. The parameters of PROCESS are the specified UDTF input parameters in the SQL statement. The input parameters are passed in as <code>Object[]</code> , and the results are output by the FORWARD function. You need to call FORWARD in the PROCESS function to determine the output data.
<code>public void close() throws UDFException</code>	The termination method of UDTF. This method is called by the framework for only once after the last record is processed.
<code>public void forward(Object ...o) throws UDFException</code>	You can call the FORWARD method to output data. Each time FORWARD is called, it outputs one record. The record corresponds to the column specified by the UDTF AS clause in the SQL statement.

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);
    }
    }
}
```


The preceding example shows how to create a UDTF in MaxCompute. If this UDTF is named `user_udtf`, you can run the following SQL statement to call this UDTF:

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
```

The values in `my_table` `col0` and `col1` are as follows:

```
+-----+-----+
| col0 | col1 |
+-----+-----+
| A B  | 1    |
| C D  | 2    |
+-----+-----+
```

The result of the `SELECT` statement is as follows:

```
+-----+-----+
| c0 | c1 |
+-----+-----+
| A  | 1  |
| B  | 1  |
| C  | 2  |
| D  | 2  |
+-----+-----+
```

6.8.5.2. UDTF description

Common uses of UDTFs in SQL:

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
select user_udtf(col0, col1) as (c0, c1) from (select * from my_table distribute by col1 sort by col1) t;
```

Notice

The following limits apply to the use of UDTF.

- No other expressions are allowed in a `SELECT` clause.

```
select col0, user_udtf(col0, col1) as (c0, c1) from mytable;
```

- UDTFs cannot be nested.

```
select user_udtf(mp_udtf(col0,col1)) as (c0,c1)from mytable;
```

UDTF examples

The user can use a UDTF to read MaxCompute resources. The following are examples of reading MaxCompute resources by using UDTFs:

1. Write UDTF program. The JAR package (`udtfexample1.jar`) is exported after compilation.

```

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 U
DTF { 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 + "|tableResource1Record
Count=" + tableResource1RecordCount + "|tableResource2RecordCount=" + tableResource2Rec
ordCount);
}
}

```

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 UDTF function (mp_udtf) in MaxCompute.

```
create function mp_udtf as com.aliyun.odps.examples.udf.UDTFResource using 'udtfexample
1.jar, file_resource.txt, table_resource1, table_resource2';
```

4. Create resource tables 'table_resource1' and 'table_resource2' in MaxCompute, and insert the corresponding data.

5. Run this UDTF.

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from table_resource1;
-- Command output:
+-----+-----+-----+
| a | b | fileResourceLineCount |
+-----+-----+-----+
| 10 | 2 | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordCount=0 |
| 10 | 2 | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordCount=0 |
+-----+-----+-----+
```

 **Note** You can also use the same method to obtain resources. For more information, see [MapReduce examples](#).

UDTF examples — Complex data types

The code in the following example defines a UDF with three overloads. The first overload uses array as the parameter; the second uses map as the parameter; and the third uses struct as the parameter. The third overload uses a struct type as the parameter or returned value, the UDF class must be supplemented with a `@Resolve` annotation to specify the specific type of struct.

```
@Resolve("struct<a:bigint>,string->string")
public class UdfArray extends UDF {
    public String evaluate(List<String> vals, Long len) {
        return vals.get(len.intValue());
    }
    public String evaluate(Map<String,String> map, String key) {
        return map.get(key);
    }
    public String evaluate(Struct struct, String key) {
        return struct.getFieldValue("a") + key;
    }
}
```

You can import a complex data type in the UDF:

```
create function my_index as 'UdfArray' using 'myjar.jar';
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as color_name from colors;
```

6.8.6. Python UDFs

6.8.6.1. Restricted environment

MaxCompute UDF uses Python V2.7. It executes user codes in a sandbox. The following operations are restricted in the sandbox:

- Read and write local files.
- Start subprocesses.
- Start threads.
- Conduct socket communication.
- Call other systems.

Due to these restrictions, user-uploaded code must all be implemented by Python, as C extension modules are disabled.

In addition, not all modules in the Python standard library are available for use. Modules that involve the preceding features are disabled. Description of available modules in the standard library:

1. All modules implemented purely by Python are available.
2. The following C extension modules are available for use.

- 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`

3. Some modules have limited functionality. For example, the sandbox limits the size that user codes can write to the standard output and standard error output. `sys.stdout` and `sys.stderr` can write up to 20 KB. Any remaining characters are ignored.

6.8.6.2. Third-party libraries

Common third-party libraries are installed in the operating environment to supplement the standard library. The supported third-party libraries include NumPy.

 **Warning** The use of third-party libraries is also subject to restrictions. For example, local or remote I/O operations are prohibited. Therefore, the related APIs in the third-party libraries are disabled.

6.8.6.3. Types of parameters and returned values

You can run the following command to specify the types of parameters and returned values:

```
@odps.udf.annotate(signature)
```

Python UDFs support the following MaxCompute SQL data types: bigint, string, double, boolean, and datetime. Before you run a SQL statement, you must specify the parameter types and returned value types of all functions. Python is a dynamically-typed language. You need to add decorators to the UDF class to specify the function signature.

The function signature is specified by a string. The syntax is as follows:

```
arg_type_list '->' type_list
arg_type_list: type_list | '*' | ''
type_list: [type_list ',' ] type
type: 'bigint' | 'string' | 'double' | 'boolean' | 'datetime'
```

Note

- The part to the left of the arrow indicates the type of parameter. The part to the right of the arrow indicates the type of returned value.
- The returned value of a UDTF can contain multiple columns. The returned value of a UDF or UDAF can contain only one column.
- * represents a variable argument. If a variable argument is specified, the UDF, UDTF, or UDAF can match any type of parameter.

Examples of valid signature:

```
'bigint,double->string'
-- The parameter is of the bigint or double type, and the returned value is of the string type.
'bigint,boolean->string,datetime'
-- The UDTF parameter is of the bigint or boolean type, and the returned value is of the string or datetime type.
'*->string'
-- Specify a variable argument: The input parameter can be of any type, and the returned value is of the string type.
'->double'
-- The parameter is NULL and the returned value is of the double type.
```

If an invalid signature is found during query parsing, an error is returned and the execution is banned. During execution, the UDF parameter with the type specified by the function signature is transferred to the user. The user returned value must be of the type specified by the function signature. Otherwise, an error is returned. The following table shows the mappings between MaxCompute SQL types and Python types.

Mapping

MaxCompute SQL type	Python type
Bigint	int
String	str
Double	float
Boolean	bool
Datetime	int

 Note

- A value of the datetime type is passed to user code as the int type. The value is the number of milliseconds that have elapsed since the epoch time. You can use the datetime module in the Python standard library to process the datetime type.
- NULL corresponds to none in Python.

In addition, the parameter of `odps.udf.int(value[, silent=True])` is modified. Parameter `silent` is added. If `silent` is true and the value cannot be converted to the int type, none is returned instead of an error.

6.8.6.4. UDFs

Implementing a Python UDF is as easy as defining a new-style class and implementing the `evaluate` method.

Example:

```
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
```



Notice A Python UDF must have its signature specified through `annotate`.

6.8.6.5. UDAFs

Description:

- `class odps.udf.BaseUDAF`: inherit this class to implement a Python UDAF.
- `BaseUDAF.new_buffer()`: implement this method and return the median 'buffer' of the aggregate function. Buffer must be mutable object (such as list and dict). The size of the buffer should not increase with the amount of data. The buffer size should not exceed 2 MB after marshal.
- `BaseUDAF.iterate(buffer[, args, ...])`: This method aggregates args into the median buffer.
- `BaseUDAF.merge(buffer, pBuffer)`: This method aggregates two median buffers; that is, aggregate pBuffer into buffer.
- `BaseUDAF.terminate(buffer)`: This method converts the median 'buffer' into the MaxCompute SQL basic types.

The following example shows how to calculate an average by using a UDAF:

```
#coding:utf-8
from odps.udf import annotate
from odps.udf import BaseUDAF
@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]
```

6.8.6.6. UDTFs

The parameters are described as follows.

Parameters

Parameter	Description
class odps.udf.BaseUDTF	Base class for a Python UDTF. Users inherit this class and implement methods such as PROCESS and CLOSE.
BaseUDTF.init()	Initialization method. To implement this method for an inherited class, you must call the initialization method super(BaseUDTF, self).init() for the base class at the beginning . The INIT method will only be called once during the entire UDTF life cycle; that is, before the first record is processed. When the UDTF needs to save internal states, all states can be initialized in this method.
BaseUDTF.process([args, ...])	The method is called by the MaxCompute SQL framework. The process method is called for each record passed in from SQL. The parameters passed into the process method are the parameters passed into the UDTF in SQL statements.
BaseUDTF.forward([args, ...])	The UDTF output method, which is called by user code. Each time FORWARD is called, one record is output. The parameters of FORWARD are the UDTF output parameters specified in SQL statements.
BaseUDTF.close()	The UDTF termination method. This method is called by the MaxCompute SQL framework. This method is called only once, 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):
    -- Output string as multiple comma-separated records.
    def process(self, arg):
        props = arg.split(',')
        for p in props:
            self.forward(p)
```



Notice A Python UDTF can also specify the parameter type or returned value type without adding 'annotate'. In this case, the function can match any input parameter in SQL. The type of returned value cannot be deduced, but all output parameters will be considered to be of the string type. Therefore, when FORWARD is called, all output values must be converted into values of the string type.

6.8.6.7. Reference resources

You can reference file and table resources in Python UDF through the `odps.distcache` module.

Syntax for referencing file resources:

```
odps.distcache.get_cache_file(resource_name)
```



Note

- Description: returns the content of the specified resource. `resource_name` is a string that corresponds to the name of an existing resource in the current project. If the resource name is invalid or does not exist, an error is returned.
- Returned value: returns file-like object. After this object is used, the caller must call the `CLOSE` method to release the resource file that is opened.

Example:

```
from odps.udf import annotate
from odps.distcache import get_cache_file
@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)
```

Command syntax:

```
odps.distcache.get_cache_table(resource_name)
```

Note

- Description: returns the content of the specified resource table. `resource_name` is a string that corresponds to the name of an existing resource table in the current project. If the resource table name is invalid or does not exist, an error is returned.
- Returned value: returns a value of the generator type. The caller traverses the table to obtain the content. Each time the caller traverses the table, a record is obtained in the form of a tuple.

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)
```

6.9. UDTs

6.9.1. Overview

User-defined types (UDTs) are introduced in MaxCompute 2.0 for the latest version of the SQL engine. UDTs allow you to reference classes or objects of third-party languages in SQL statements to obtain data or call methods.

UDTs are typically applied in the following scenarios:

- **Scenario 1:** MaxCompute does not have built-in functions to complete tasks that can be easily performed using other languages. For example, there are some tasks that can be performed by calling a single built-in Java class. Performing these tasks with user defined functions (UDFs) is complex.
- **Scenario 2:** You need to call a third-party library in SQL statements to implement the corresponding feature. You want to use a feature provided by a third-party library directly in a SQL statement, instead of wrapping the feature inside a UDF.
- **Scenario 3:** SELECT TRANSFORM allows you to include objects and classes in SQL statements to make these SQL statements easier to read and maintain. For some languages, such as Java, the source code can be only executed after it is compiled. You want to reference objects and classes of these languages in SQL statements.

Notice

- UDTs only support Java.
- All operators use the semantics of MaxCompute SQL.
- UDTs cannot be used as shuffle keys in the JOIN, GROUP BY, DISTRIBUTE BY, SORT BY, ORDER BY, and CLUSTER BY clauses.
- DDL statements do not support UDTs. You cannot create tables that contain UDT objects. The final output cannot be UDT types.

6.9.2. Feature summary

UDTs allow you to reference classes or objects of third-party languages in SQL statements to obtain data or call methods.

The UDTs supported in MaxCompute are very different from those in other SQL engines.

UDTs supported by other SQL engines are similar to the struct composite type in MaxCompute. UDTs supported by MaxCompute are similar to the CREATE TYPE statement. A UDT contains both fields and methods. Additionally, MaxCompute does not require that you use Data Definition Language (DDL) statements to define type mappings. MaxCompute allows you to reference types directly in SQL statements.

Example:

```
set odps.sql.type.system.odps2=true;
SELECT Integer.MAX_VALUE;
-- A similar output is displayed:
+-----+
| max_value |
+-----+
| 2147483647 |
+-----+
```

The expression in the preceding SELECT statement is similar to a Java expression and executed in the same manner as it would in Java. The expression specifies a UDT in MaxCompute.

You can use UDFs to implement all features provided by UDTs, but with some complexity. If you use a UDF to implement the same feature, you need to follow these steps:

1. 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;
    }
}
```

2. Compile the UDF as a JAR package. Upload the JAR 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';
```

3. Call the function in a SQL statement.

```
select integer_max_value();
```

A UDT simplifies this procedure. By using UDTs, you can use features provided by other languages in SQL statements.

6.9.3. Feature description

The example described in the Feature overview topic demonstrates how to use user-defined types (UDTs) to access the static fields of Java classes. UDTs can be used to implement a number of functions. The following example shows a UDT execution procedure and its features.

```
-- Sample data
@table1 := select * from values ('10000000000000000000') as t(x);
@table2 := select * from values (100L) as t(y);
-- Code logic
@a := select new java.math.BigInteger(x) x from @table1;           -- Create an object by using the new method.
@b := select java.math.BigInteger.valueOf(y) y from @table2;       -- Call a static method.
select /*+mapjoin(b)*/ x.add(y).toString() from @a a join @b b;   -- Call an instance method.
```

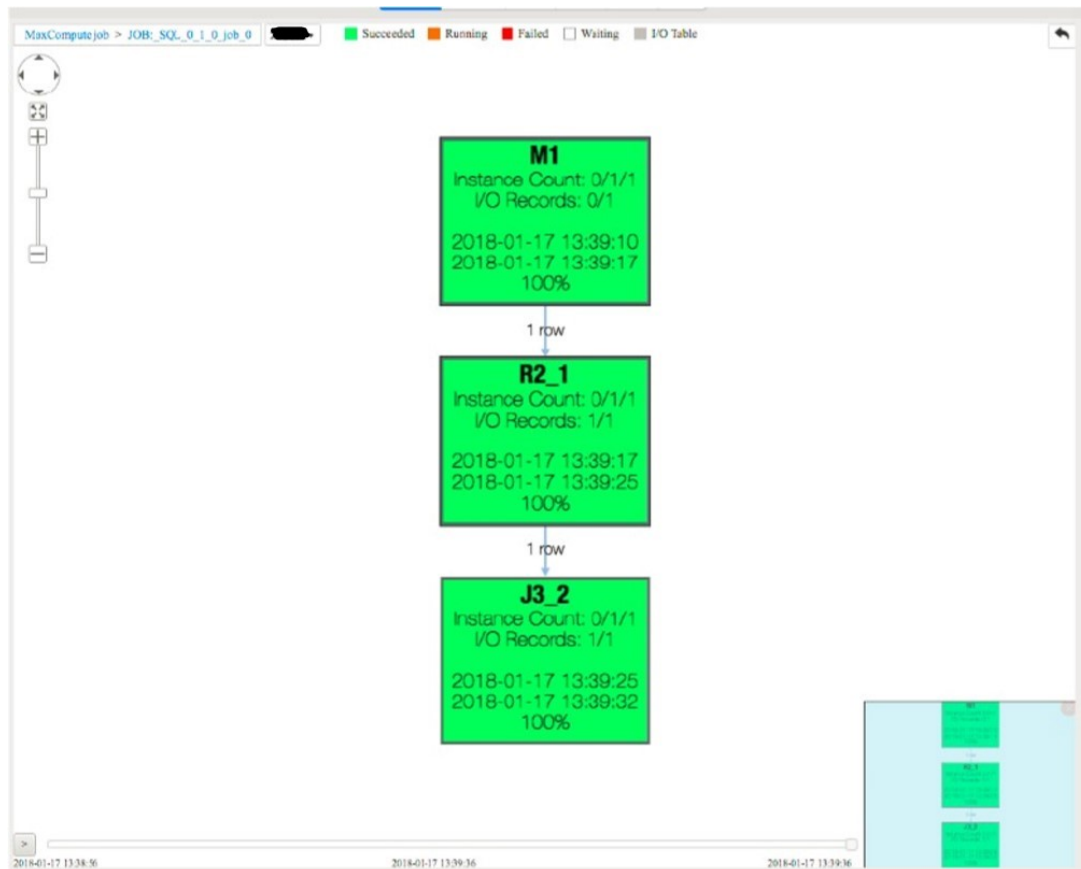
The following result is returned:

```
1000000000000000000100
```

 **Note** This example also shows how to use subqueries with UDT columns. User-defined functions (UDFs) cannot be used in such subqueries. Variable a in the x column is of the java.math.BigInteger class, not a built-in class. You can pass UDT data to another operator and then call the required method. You can also use UDT data in data shuffling.

UDT execution

Example



The preceding figure shows the three stages of a UDT: M1, R2, and J3. 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 called at the J3 stage.

If a JOIN clause is used in MapReduce, data must be reshuffled. As a result, data is processed at multiple stages. Data is processed at different stages or even by different processes or physical machines. The UDT encapsulates these stages and functions as a JVM.

Description

- UDTs support only Java.
- 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`.

Example:

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.resources=odps-test.jar; -- Specify the JAR package that you want
to reference. Before you reference the JAR package, upload the package to your project.
select new com.aliyun.odps.test.IntegerMaxValue().evaluate();
```

 **Notice** This flag is the same as the flag that is used to specify resources in the SELECT TRANSFORM statement. Therefore, this flag affects JAR package uploading in UDTs and resource settings in the SELECT TRANSFORM statement.

- **odps.sql.session.java.imports:** specifies the default Java package. Separate multiple Java packages with commas (.). This flag is similar to the IMPORT statement in Java. You can specify a classpath, such as `java.math.BigInteger`, or use `*`. Static import is not supported.

Example:

```
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 Java
package.
select new IntegerMaxValue().evaluate();
```

- UDTs allow you to:
 - Create objects by using the new method.
 - Create arrays by using the new method, including ArrayList initialization. Example: `new Integer[] { 1, 2, 3 }`.
 - Call methods, including static methods. You can create objects in the factory method pattern.
 - Access fields, including static fields.

-  **Notice**
- Identifiers in UDTs contain package names, class names, method names, and field names. All identifiers are case-sensitive.
 - UDTs support SQL type conversions, such as `cast(1 as java.lang.Object)`. UDTs do not support Java type conversions, such as `(Object)1`.
 - Anonymous classes and lambda expressions are not supported.
 - Functions that do not return values cannot be called in UDTs.

 **Note** UDTs are used in expressions. Functions that do not return values cannot be called in expressions.

- All SDK for Java classes can be referenced by UDTs. The JDK runtime environment is JDK 1.8. Later versions may not be supported.
- All operators use the semantic of MaxCompute SQL. 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.

You may be confused about the role of the `=` operator. The `=` operator in SQL statements is used as a comparison operator. It is used to compare one expression with another expression. You must call the equals method in Java to compare whether two objects are equivalent. The `=` operator cannot be used to verify the equivalence of two objects.

- Java data types are mapped to built-in data types. The mapping can be applied to UDTs.
 - 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 CHR built-in function 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 are applied.

 **Notice** 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.

- UDTs completely support Java generics. For example, based on the parameter type, the compiler can determine that the value returned by `java.util.Arrays.asList(new java.math.BigInteger('1'))` is `java.util.List<java.math.BigInteger>` .

 **Notice** You must specify the type parameter in a constructor function or use `java.lang.Object`. This is the same as Java. For example, the result of `new java.util.ArrayList(java.util.Arrays.asList('1', '2'))` is of the `java.util.ArrayList<Object>` type. The result of `new java.util.ArrayList<String>(java.util.Arrays.asList('1', '2'))` is of the `java.util.ArrayList<String>` type.

- UDTs do not have a clear definition of **object equality**. This is caused by data reshuffling. The JOIN example shows that objects may be transmitted between different processes or physical machines. During 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 equate two objects.

 **Note** 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 in expressions, but cannot be used as outputs. For example, you cannot call the `group by new java.math.BigInteger('123')` method. However, you can call the `group by new java.math.BigInteger('123').hashCode()` method. This is because the value returned by hashCode is an int.class type, which can be used as the built-in INT type.

- The following type conversion rules are extended 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 for two objects without inheritance follows native conversion rules.

 **Notice** The conversion may cause data changes. 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 cause data changes or even data precision loss.

- UDT objects cannot be saved or added to tables. DDL statements do not support UDTs. You cannot create tables that contain UDT objects unless the data type is implicitly converted to one of the built-in types. In addition, the output cannot be a UDT. However, you can call the `toString()` method to convert the data type to the `java.lang.String` type because the `toString()` method supports all Java classes. You can use this method to check UDT data during debugging.

You can also add the `set odps.sql.udt.display.toString=true;` flag to enable MaxCompute to convert all output UDT data to strings by calling the `java.util.Objects.toString(...)` method for debugging.

 **Note** This flag is typically used for debugging because it can be applied only to `PRINT` statements. It cannot be applied to `INSERT` statements.

`BINARY` is a built-in type and supports automatic serialization. You can save the `byte[]` arrays. The saved `byte[]` arrays can be deserialized to the `BINARY` type.

Some classes may have their own serialization and deserialization methods, such as `protobuf`. To save UDTs, you must call serialization and deserialization methods to convert the data type to `BINARY`.

- You can use UDTs to achieve the feature provided by the `SCALAR` function. You can use the `COLLECT_LIST` and `EXPLODE` built-in functions with UDTs to achieve the features provided by aggregate and table-valued functions.
- UDTs support resource access. You can call the `com.aliyun.odps.udf.impl.UDTExecutionContext.get()` static method to obtain the `ExecutionContext` object. Then, use the object to access the current execution context and then to access resources, such as files and tables.

6.9.4. More examples

6.9.4.1. Example of using Java arrays

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}, -- Create an array that contains three elements by initial
    izing an ArrayList.
    new Integer[][] { new Integer[] {c1, c2}, new Integer[] {c3, c4} }, -- Create a multid
    imensional array.
    new Integer[] {c1, c2, c3} [2], -- Access the elements in the array using indexes.
    java.util.Arrays.asList(c1, c2, c3); -- This is another way to create a built-in arr
    ay. It creates a List<Integer>, which can be used as an array<int>.
from values (1,2,3,4) as t(c1, c2, c3, c4);

```

6.9.4.2. Example of using JSON

The runtime of UDT carries a GSON dependency (version 2.2.4), which can be directly used in GSON.

Example:

```

set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*,java.com.google.gson.*; -- To import multipl
e packages, 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 str
    ing. GSON also forcibly converts the deserialized result from List<Object> type to List<Str
    ing> type.
from @a;

```

Compared with built-in function `GET_JSON_OBJECT`, this method is simple and improves efficiency by extracting content from the JSON string and deserializing the string to a supported data type.

In addition to GSON 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).

6.9.4.3. Example of using composite types

Built-in types of array and map are mapped to `java.util.List` and `java.util.Map`, respectively.

- Java objects in classes calling the `java.util.List` or `java.util.Map` API can be used in MaxCompute SQL composite type data processing.
- Array and map type data in MaxCompute can directly call the `java.util.List` or `java.util.Map` API.

Example:

```

set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*;
select
    size(new ArrayList<Integer>()),          -- Call built-in function size to obtain the size of the ArrayList.
    array(1,2,3).size(),                    -- Call the List method for built-in type array.
    sort_array(new ArrayList<Integer>()),    -- Sort the data in the ArrayList.
    al[1],                                  -- The Java List method does not support indexing. However, the array type supports indexing.
    Objects.toString(a),                    -- With this method, you can convert array type to string type data.
    array(1,2,3).subList(1, 2)              -- Get a sublist.
from (select new ArrayList<Integer>(array(1,2,3)) as al, array(1,2,3) as a) t;

```

6.9.4.4. Example of aggregation

To achieve aggregation with UDTs, you must first use built-in function `COLLECT_SET` or `COLLECT_LIST` to convert the data to the List type and then call the UDT methods to aggregate the data.

The following example shows how to obtain the median from `BigInteger` data. You cannot directly call the built-in `MEDIAN` function because the data is `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; -- To obtain the median, first 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;
-- Final output.
select med.toString() from @c;

```

You cannot use the `COLLECT_LIST` function to implement partial aggregation because it aggregates all data. It is more efficient to use the built-in aggregator or UDAF object. We recommend that you use the built-in aggregator. Aggregating all data in a group increases the risk of data skew.

If the logic of the UDAF object is to aggregate all data in a similar manner to built-in function `WM_CONCAT`, using the `COLLECT_LIST` function is more efficient than using the UDAF object.

6.9.4.5. Example of using table-valued functions

Table-valued functions allow you to input and output multiple rows and columns. To input or output multiple rows and columns, follow these steps:

1. For more information about how to input multiple rows or columns, see the example of using aggregate functions.
2. To output multiple rows, you can use a UDT to define a Collection type (List or Map), and then call the `EXPLODE` function to split the collection into multiple rows.
3. A UDT can contain multiple fields. You can retrieve the data from the fields by calling different getter methods. The data is then output in multiple rows.

The following example shows how to split a JSON string and output the result as multiple columns:

```
@a := select ' [{"a": "1", "b": "2"}, {"a": "1", "b": "2"} ]' str; -- 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; -- Output the splitting result in multiple columns.
select a.toString() a, b.toString() b from @d; -- The final output. Columns a and b in variable d are of the Object type.
```

6.9.5. Feature advantages

UDT has the following features:

- Easy to use. You do not need to define any functions.
- To improve the flexibility of SQL, all JDK supported features can be used directly.
- You can directly reference objects and classes of other languages in SQL statements.
- You can directly reference the libraries of other language and reuse code that you have written in other languages.
- You can create object-oriented features.

6.9.6. Performance advantages

UDTs and UDFs use similar execution procedures and provide similar performance. However, UDTs have higher performance in certain scenarios where the compute engine has been greatly improved.

- Deserialization is not required for objects in only one process. Deserialization is required only when the objects are transmitted among processes. This means that UDT do not incur any serialization or deserialization overhead when no data reshuffling is performed, such as calling the join or aggregator function.
- UDTs suffer no performance loss from reflection because the runtime of UDTs is based on Codegen, rather than based on reflection.
- Multiple UDTs can be wrapped into a single function call and executed together. In the following example, a single UDT is being called. UDTs focus on small-granularity data processing. This does not incur additional overhead for the API where multiple functions are called.

```
values[x].add(values[y]).divide(java.math.BigInteger.valueOf(2))
```

6.9.7. Security advantages

UDTs are restricted in the Java sandbox model similar to UDFs. To perform restricted operations, you must enable sandbox isolation or apply to join the sandbox whitelist.

6.10. UDJ

6.10.1. Overview

MaxCompute provides multiple JOIN methods natively, including INNER JOIN, RIGHT JOIN, OUTER JOIN, LEFT JOIN, FULL JOIN, SEMIJOIN, and ANTISEMIJOIN methods. You can use these native JOIN methods in most scenarios. However, these methods cannot handle multiple tables.

In most cases, you can build your code framework using UDFs. However, the current UDF, UDTF, and UDAF frameworks only can handle one table at a time. To perform user-defined operations for multiple tables, you have to use native JOIN methods, UDFs, UDTFs, and complex SQL statements. In certain cases when you handle multiple tables, you must use a custom MapReduce framework instead of SQL to complete the required task.

In any situation, these operations require technological expertise and may cause the following problems:

- Calling multiple JOIN methods in SQL statements can lead to computational black box that is complex and difficult to execute with minimal overheads.
- Using MapReduce even make optimal execution of code becomes impossible. Most of the MapReduce code is written in Java. The execution of the MapReduce code is less efficient than the execution of MaxCompute code generated by the LLVM code generator at an optimized native runtime.

With the addition of the MaxCompute 2.0 compute engine, the user defined join (UDJ) API has been added to the user defined function (UDF) framework. This API allows you to handle multiple tables and simplifies operations performed in the underlying MapReduce distributed system.

6.10.2. UDJ usage

6.10.2.1. Examples

The following example describes how to use UDJ in MaxCompute.

This example uses the payment table and the user_client_log table.

- The payment (user_id string,time datetime,pay_info string) table stores the payment information of a user. Each payment record includes the user ID, payment time, and the payment details.
- The user_client_log (user_id string,time datetime,content string) table stores user client records, including the user ID, operation time, and operation.

Requirements: For each record in the user_client_log table, locate the payment record that has the time closest to the operation time, and join and output the content of both records.

To complete this task by using standard join methods, you would need to join the two tables based on their common user_id fields, and then locate the payment record and operation that most closely match each other's time. The SQL statement may be written as follows:

```
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))
```

However, when 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, and the aggregate function cannot be called in the join condition. Because of this, this task cannot be completed by calling the standard JOIN method.

Can we use UDJ to solve this problem? Yes. The following topics describe how to use UDJ to satisfy the preceding requirements.

6.10.2.2. Use Java to write the UDJ code

Prerequisites

UDJ is a new feature, so a new SDK is required.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.30.0</version>
  <scope>provided</scope>
</dependency>
```

The SDK contains a new abstract class UDJ. All UDJ features can be implemented through this class.

Sample code

The following sample code is used for reference only.

```
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 of right table, find the nearest record of left table and
 * merge two records.
 */
@Resolve("->string,bigint,string")
public class PayUserLogMergeJoin extends UDJ {
    private Record outputRecord;
    /** Will be called prior to the data processing phase. User could implement
     * this method to do initialization work.
     */
    @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)
        });
    }
    /** Override this method to implement join logic.
     * @param key Current join key
     * @param left Group of records of left table corresponding to the current key
     * @param right Group of records of right table corresponding to the current key
```

```

    * @param output Used to output 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()) {
            // Empty right group, do nothing.
            return;
        } else if (! left.hasNext()) {
            // Empty left group. Output all records of right group without merge.
            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 left group of records will be iterated from the start to the end
        // for each record of right group, but the iterator cannot be reset.
        // So we save every records of left to an 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 through all records of left, and find the pay record that has
            // the minimal difference in terms of time.
            for (Record pay: pays) {
                long delta = Math.abs(logTime - pay.getDatetime(0).getTime());
                if (delta < minDelta) {
                    minDelta = delta;
                    nearestPay = pay;
                }
            }
            // Merge the log record with nearest pay record and output to the result.
            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() {
    }
}

```

 **Notice** In this example, the NULL values in the entries are not processed. To simplify the data processing procedure, assume that no NULL values are contained in the tables.

Each time you call this JOIN method of UDJ, records that match the same key in the two tables are returned. Therefore, UDJ searches all records in the payment table to locate the record with the time closest to each record in the user_client_log table.

Assume that the user only has a few payment records. In this case, you can load the data in the payment table to the memory. Typically, there is sufficient memory to store the user payment data generated each day. What if this assumption is invalid? How can we resolve this issue? This issue will be discussed in **Pre-sorting**.

6.10.2.3. Create a UDJ function in MaxCompute

After you have written the UDJ code in Java, upload the code to MaxCompute SQL as a plug-in. You must have registered the code with MaxCompute first.

Assume that the code is compressed into JAR package odps-udj-example.jar. Use the Add JAR command to upload the JAR package to MaxCompute.

```
add jar odps-udj-example.jar;
```

Execute the CREATE FUNCTION statement to create UDJ function *pay_user_log_merge_join*, using JAR package odps-udj-example.jar and Java class *com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin*.

```
create function pay_user_log_merge_join
as 'com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin'
using 'odps-udj-example.jar';
```

6.10.2.4. Use UDJ in MaxCompute SQL

After you have registered UDJ in the database, UDJ can be used in MaxCompute SQL.

1. Create a sample source table.

```
create table payment (user_id string,time datetime,pay_info string);
create table user_client_log(user_id string,time datetime,content string);
```

2. Create sample data.

 **Notice** The data in this example is only used for reference. You may need to create different data in actual operations.

```

-- Create data in 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')
;

-- Create data in 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 OkTYNUHMqZzlDyL'),
('8881237', datetime '2018-02-13 10:30:00', 'click OkTYNUHMqZzlDyL'),
('9890100', datetime '2018-02-13 16:01:00', 'click vOTQfBFjcgXisYU'),
('9890100', datetime '2018-02-13 16:20:00', 'click WxaLgOCcVEvhiFJ')
;

```

3. In MaxCompute SQL, use the UDJ function you have created:

```

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)
;

```

 **Note** The syntax of UDJ is similar to that of the standard JOIN statement. The only difference is that the USING clause is added to UDJ.

Description:

- **pay_user_log_merge_join** is the name of the UDJ function in SQL.
- **(p.time, p.pay_info, u.time, u.content)** are the columns used in these two tables.

- **r** is 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.
4. Execute this SQL statement. A similar output is displayed:

```
+-----+-----+-----+
| 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 gbAfPCwrGXvEjpI, 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 |
+-----+-----+-----+
```

As shown in the preceding code, the task that could not be performed by calling native JOIN methods has been completed by using UDJ.

6.10.2.5. Pre-sorting

An iterator is used to search all records in the payment table and locate payment records that match the query. 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 RAM size limits, 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 using the **SORT BY** clause. When the size of the payment data is too large to be stored in the memory, it would be easier to address this issue if all data in the table has already been sorted by time. You then only need to compare the first element in these two lists. UDJ code in Java:

```

@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 delta of time between two records is decreasing, we can still
            // explore the left group to try to gain a smaller delta.
            lastPayRecord = payRecord.clone();
            prevDelta = delta;
            payRecord = left.next();
        } else {
            // Hit to the point of minimal delta. Check with the last pay record,
            // output the merge result and prepare to process the next record of
            // right group.
            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;
            }
        }
    }
}

```



Notice After you have modified the UDJ code, you must update the corresponding JAR package.

When the created UDJ function is used in MaxCompute SQL, you must modify the command as follows:

```

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)
SORT BY p.time, u.time
;

```

In the native SQL language, you must make a few modifications, add a SORT BY clause to the end of the UDJ clause, and then sort the data in both tables by time.

The execution result is the same as the result before the code is modified.

This method uses the SORT BY clause to pre-sort the data. To achieve the same result, only a maximum of three records need to be cached.

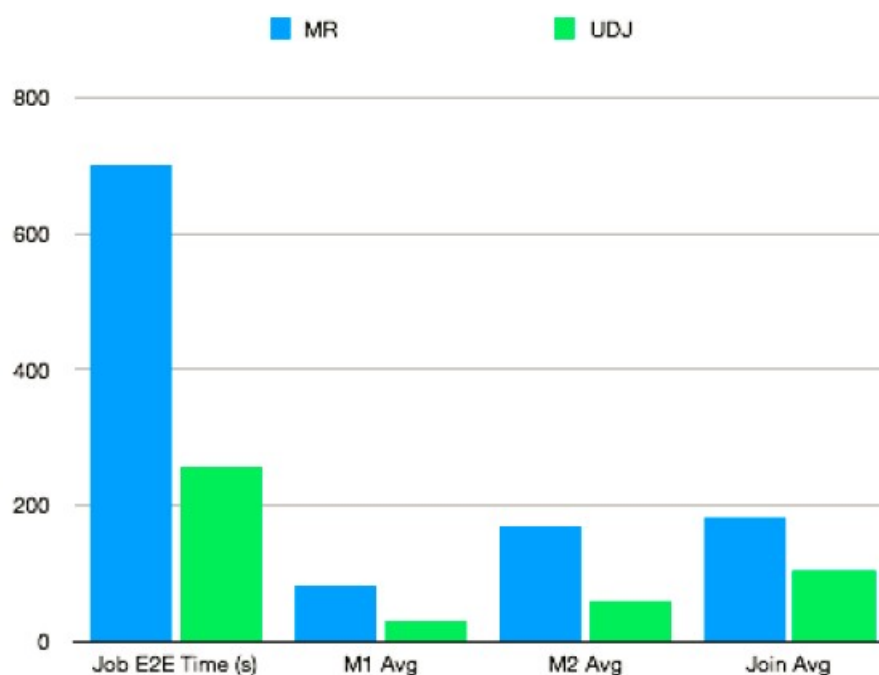
6.10.3. Performance advantages

Without UDJ, you must use MapReduce to handle complex cross-table computing tasks in a distributed system.

The following example uses an online MapReduce job to test the UDJ performance. This MapReduce job uses a complex algorithm to join two tables. This example uses UDJ to rewrite the SQL statements of the MapReduce job and checks the execution results.

Under the same programming concurrency, the comparison of performance is as follows.

Performance comparison



As shown in the figure, UDJ helps describe the complex logic of handling multiple tables, and greatly improves the query performance.

Note The code is only executed inside UDJ. The entire logic of the code is executed by the high-performance MaxCompute native runtime.

UDJ optimizes the MaxCompute runtime engine and the data exchange between interfaces. The join logic of UDJ is more efficient than that of the reduce stage.

6.11. Parameterized view

MaxCompute supports parameterized views. You can call such views to obtain queried information.

In the traditional views of MaxCompute, complex SQL scripts are encapsulated at the underlying layer. Callers can call views like reading a standard table without the need to understand the underlying implementation mechanism. Traditional views are widely used because they can be used to implement encapsulation and code reuse.

However, you cannot specify parameters for traditional views. If a traditional view is called to read data from an underlying table, you cannot filter data in the underlying table or pass other parameters to the view. This reduces the code reuse rate.

The new SQL engine of MaxCompute V2.0 supports parameterized views and allows you to import any tables or other variables to customize views.

Create a parameterized view

Example

```
--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;
```

- The created pv1 view has two parameters, the table and string parameters. The parameter values can be tables or be of a basic data type.
- The parameter values can also be subqueries. Example:

```
select * from view_name( (select 1 fromsrc where a > 0), 1);
```

- When you define a view, you can set the type of a parameter value to ANY. Example:

```
create view view_name (@a ANY, @b TABLE (x ANY)) as ...
```

- When you define a view, you can use an asterisk (*) to indicate a varying-length column. Example:

```
create view view_name(@a bigint @b TABLE(x bigint, * ANY)) asselect * from @b where x = @a;
```

 **Note** In the table specified by the TABLE parameter, data in the first column is of the BIGINT type. You can execute `SELECT *` to obtain the varying-length part.

Call the created view

Execute the following statement to call the created pv1 view:

```
@a := select * from src where value >0;
--call view with table variable an 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 a physical table, view, table variable, or table alias in common table expressions (CTEs).
- Common parameters can be variables or constants.

Additional instructions

A parameterized view can contain multiple SQL statements, similar to a script.

```
--view with parameters
-- param @a -a table parameter
-- param @b -a string parmeter
--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

- Content between BEGIN and end is the script of this view.
- The `@pv2 := ...` statement is similar to the RETURN statement in other programming languages. This statement is used to assign a value to an implicit table variable that has the same name as the view.
- Only DML statements can be used in scripts. The INSERT and CREATE TABLE AS statements cannot be included in scripts.
- PRINT statements cannot be included in scripts.

The matching rules for actual and formal view parameters are the same as those specified in a normal programming language. If view parameters can be implicitly converted, these parameters can be matched. For example, the BIGINT value can match the 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 parameter that has the same schema as Table B.

In some situations, you can declare the return type to make the code easier to read. Example:

```
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 is TABLE (x string, y string), which indicates the type returned to the caller. You can use this parameter to customize the table schema.
- The response parameter is @ret. A value is assigned to the parameter in the view script.
- You can regard a view that does not contain the BEGIN and END keywords or that does not return variables as a simplified view.

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 `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

Function declaration:

```
ST_AsBinary(ST_Geometry)
```

Description: This function returns the well-known binary (WKB) representation of the input geometry.

Example:

```
SELECT ST_AsBinary(ST_Point(1, 2)) FROM onerow;  
-- WKB representation of POINT (1 2)
```

6.12.2.2. ST_AsGeoJson

Function declaration:

```
ST_AsGeoJson(geometry)
```

Description: This function returns the GeoJSON representation of the input geometry.

Example:

```
SELECT ST_AsGeoJson(ST_Point(1.0, 2.0)) from onerow;  
-- {"type":"Point", "coordinates":[1.0, 2.0]}
```

6.12.2.3. ST_AsJson

Function declaration:

```
ST_AsJSON(ST_Geometry)
```

Description: This function returns the JSON representation of the input geometry.

Example:

```
SELECT ST_AsJSON(ST_Point(1.0, 2.0)) from onerow;  
-- {"x":1.0,"y":2.0}  
SELECT ST_AsJSON(ST_SetSRID(ST_Point(1, 1), 4326)) from onerow;  
-- {"x":1.0,"y":1.0,"spatialReference":{"wkid":4326}}
```

6.12.2.4. ST_AsShape

Function declaration:

```
ST_AsShape(ST_Geometry)
```

Description: This function returns the ESRI shape representation of the input geometry.

Example:

```
SELECT ST_AsShape(ST_Point(1, 2)) FROM onerow;  
-- Esri shape representation of POINT (1 2)
```

6.12.2.5. ST_AsText

Function declaration:

```
ST_AsText(ST_Geometry)
```

Description: This function returns the well-known text (WKT) representation of the input geometry.

Example:

```
SELECT ST_AsText(ST_Point(1, 2)) FROM onerow;  
-- POINT (1 2)
```

6.12.2.6. ST_GeomCollection

Function declaration:

```
ST_GeomCollection(wkt)
```

Description: This function constructs a multi-part geometry from the well-known text (WKT) representation based on 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)')  
-- Not supported.
```

6.12.2.7. ST_GeomFromGeoJson

Function declaration:

```
ST_GeomFromGeoJson(json)
```

Description: This function constructs a geometry from the input GeoJSON representation.

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 L
IMIT 1;
-- Construct a linestring.
```

6.12.2.8. ST_GeomFromJSON

Function declaration:

```
ST_GeomFromJSON(json)
```

Description: This function constructs a geometry from the input ESRI JSON representation.

Example:

```
SELECT ST_GeomFromJSON('{"x":0.0,"y":0.0}') FROM src LIMIT 1;
-- Construct a point.
```

6.12.2.9. ST_GeomFromShape

Function declaration:

```
ST_GeomFromShape(shape)
```

Description: This function constructs a geometry from the input ESRI shape representation.

Example:

```
SELECT ST_GeomFromShape(ST_AsShape(ST_Point(1, 2)));
-- Construct a point.
```

6.12.2.10. ST_GeomFromText

Function declaration:

```
ST_GeomFromText(wkt)
```

Description: This function constructs a geometry from the input well-known text (WKT) representation based on 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

Function declaration:

```
ST_GeomFromWKB(wkb)
```

Description: This function constructs a geometry from the input well-known binary (WKB) representation based on the Open Geospatial Consortium (OGC).

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

Function declaration:

```
ST_GeometryType(geometry)
```

Description: This function returns the type name of the input geometry.

Example:

```
SELECT ST_GeometryType(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- ST_Point
SELECT ST_GeometryType(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
-- ST_LineString
SELECT ST_GeometryType(ST_Polygon(2,0, 2,3, 3,0)) FROM src LIMIT 1;
-- ST_Polygon
```

6.12.2.13. ST_LineString

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

Function declaration:

```
ST_LineFromWKB(wkb)
```

Description: This function constructs a two-dimensional line from the input well-known binary (WKB) representation based on 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

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;
```

6.12.2.16. ST_MLineFromWKB

Function declaration:

```
ST_MLineFromWKB(wkb)
```

Description: This function constructs a two-dimensional multilinestring from the input well-known binary (WKB) representation based on the Open Geospatial Consortium (OGC).

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

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

Function declaration:

```
ST_MPointFromWKB(wkb)
```

Description: This function constructs a two-dimensional multipoint geometry from the input well-known binary (WKB) representation based on 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

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 s
rc LIMIT 1;
SELECT ST_MultiPolygon('multipolygon (((0 0, 0 1, 1 0, 0 0)), ((2 2, 2 3, 3 2, 2 2)))') fro
m src LIMIT 1;
```

6.12.2.20. ST_MPolyFromWKB

Function declaration:

```
ST_MPolyFromWKB(wkb)
```

Description: This function constructs a two-dimensional multipolygon from the input well-known binary (WKB) representation based on the Open Geospatial Consortium (OGC).

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

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;
```

6.12.2.22. ST_PointFromWKB

Function declaration:

```
ST_PointFromWKB(wkb)
```

Description: This function constructs a two-dimensional point from the input well-known binary (WKB) representation based on the Open Geospatial Consortium (OGC).

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

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;
```

6.12.2.24. ST_Polygon

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

Function declaration:

```
ST_PolyFromWKB(wkb)
```

Description: This function constructs a two-dimensional polygon from the input well-known binary (WKB) representation based on 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

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

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;
-- 9.0
```

6.12.3.2. ST_Centroid

Function declaration:

```
ST_Centroid(polygon)
```

Description: This function returns the center point of the minimum bounding rectangle of the input polygon.

Example:

```
SELECT ST_Centroid(ST_GeomFromText('polygon ((0 0, 3 6, 6 0, 0 0))')) FROM src LIMIT 1;
-- POINT(3 3)
SELECT ST_Centroid(ST_GeomFromText('polygon ((0 0, 0 8, 8 0, 0 0))')) FROM src LIMIT 1;
-- POINT(4 4)
```

6.12.3.3. ST_CoordDim

Function declaration:

```
ST_CoordDim(geometry)
```

Description: This function returns the coordinate dimension of the input geometry.

Example:

```
SELECT ST_CoordDim(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- 2
SELECT ST_CoordDim(ST_PointZ(1.5,2.5, 3)) FROM src LIMIT 1;
-- 3
SELECT ST_CoordDim(ST_Point(1.5, 2.5, 3., 4.)) FROM src LIMIT 1;
-- 4
```

6.12.3.4. ST_Dimension

Function declaration:

```
ST_Dimension(geometry)
```

Description: This function returns the spatial dimension of the input geometry.

Example:

```
SELECT ST_Dimension(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- 0
SELECT ST_Dimension(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
-- 1
SELECT ST_Dimension(ST_Polygon(2,0, 2,3, 3,0)) FROM src LIMIT 1;
-- 2
```

6.12.3.5. ST_Distance

Function declaration:

```
ST_Distance(ST_Geometry1, ST_Geometry2)
```

Description: This function returns the distance between a point in geometry1 and a point in geometry2.

Example:

```
SELECT ST_Distance(ST_Point(0.0,0.0), ST_Point(3.0,4.0)) FROM src LIMIT 1;
-- 5.0
```

6.12.3.6. ST_GeodesicLengthWGS84

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.

Example:

```
SELECT ST_GeodesicLengthWGS84(ST_SetSRID(ST_LineString(0.0,0.0, 0.3,0.4), 4326)) FROM src L
IMIT 1;
-- 55km
SELECT ST_GeodesicLengthWGS84(ST_GeomFromText('MultiLineString((0.0 80.0, 0.3 80.4))', 4326
)) FROM src LIMIT 1;
-- 45km
```

6.12.3.7. ST_GeometryN

Function declaration:

```
ST_GeometryN(ST_GeometryCollection, n)
```

Description: This function returns the nth geometry in the input geometry collection. n starts from 1.

Example:


```
SELECT ST_GeometryN(ST_GeomFromText('multipoint ((10 40), (40 30), (20 20), (30 10))'), 3)
FROM src LIMIT 1;
-- ST_Point(20 20)
SELECT ST_GeometryN(ST_GeomFromText('multilinestring ((2 4, 10 10), (20 20, 7 8))'), 2) FROM
M src LIMIT 1;
-- ST_Linestring(20 20, 7 8)
```

6.12.3.8. ST_Is3D

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
SELECT ST_Is3D(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- false
SELECT ST_Is3D(ST_Point(3., 4.)) FROM src LIMIT 1;
-- false
SELECT ST_Is3D(ST_PointZ(3., 4., 2)) FROM src LIMIT 1;
-- true
```

6.12.3.9. ST_IsClosed

Function declaration:

```
ST_IsClosed(ST_[Multi]LineString)
```

Description: If the input linestring or linestrings are closed, this function returns true.

Example:

```
SELECT ST_IsClosed(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- true
SELECT ST_IsClosed(ST_LineString(0.,0., 3.,4.)) FROM src LIMIT 1;
-- false
```

6.12.3.10. ST_IsEmpty

Function declaration:

```
ST_IsEmpty(geometry)
```

Description: If the input geometry is empty, this function returns true.

Example:

```
SELECT ST_IsEmpty(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- false
SELECT ST_IsEmpty(ST_GeomFromText('point empty')) FROM src LIMIT 1;
-- true
```

6.12.3.11. ST_IsMeasured

Function declaration:

```
ST_IsMeasured(geometry)
```

Description: If the input geometry has M coordinates (measures), this function returns true.

Example:

```
SELECT ST_IsMeasured(ST_Polygon(1,1, 1,4, 4,4, 4,1)) FROM src LIMIT 1;
-- false
SELECT ST_IsMeasured(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- false
SELECT ST_IsMeasured(ST_Point(3., 4.)) FROM src LIMIT 1;
-- false
SELECT ST_IsMeasured(ST_PointM(3., 4., 2)) FROM src LIMIT 1;
-- true
```

6.12.3.12. ST_IsSimple

Function declaration:

```
ST_IsSimple(geometry)
```

Description: If the input geometry is simple, this function returns true.

Example:

```
SELECT ST_IsSimple(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- true
SELECT ST_IsSimple(ST_LineString(0.,0., 1.,1., 0.,1., 1.,0.)) FROM src LIMIT 1;
-- false
```

6.12.3.13. ST_IsRing

Function declaration:

```
ST_IsRing(ST_LineString)
```

Description: If the input linestring is closed or simple, this function returns true.

Example:

```
SELECT ST_IsRing(ST_LineString(0.,0., 3.,4., 0.,4., 0.,0.)) FROM src LIMIT 1;
-- true
SELECT ST_IsRing(ST_LineString(0.,0., 1.,1., 1.,2., 2.,1., 1.,1., 0.,0.)) FROM src LIMIT 1;
-- false
SELECT ST_IsRing(ST_LineString(0.,0., 3.,4.)) FROM src LIMIT 1;
-- false
```

6.12.3.14. ST_Length

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;
-- 5.0
```

6.12.3.15. ST_M

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;
-- 2
```

6.12.3.16. ST_MaxM

Function declaration:

```
ST_MaxM(geometry)
```

Description: This function returns the maximum M coordinate of the input geometry.

Example:

```
SELECT ST_MaxM(ST_PointM(1.5, 2.5, 2)) FROM src LIMIT 1;
-- 2
SELECT ST_MaxM(ST_LineString('linestring m (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
-- 1
```

6.12.3.17. ST_MinM

Function declaration:

```
ST_MinM(geometry)
```

Description: This function returns the minimum M coordinate of the input geometry.

Example:

```
SELECT ST_MinM(ST_PointM(1.5, 2.5, 2)) FROM src LIMIT 1;
-- 2
SELECT ST_MinM(ST_LineString('linestring m (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
-- 1
```

6.12.3.18. ST_X

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;
-- 1.5
```

6.12.3.19. ST_Y

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;
-- 2.5
```

6.12.3.20. ST_Z

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;
-- 1.5
```

6.12.3.21. ST_MaxX

Function declaration:

```
ST_MaxX(geometry)
```

Description: This function returns the maximum X coordinate of the input geometry.

Example:

```
SELECT ST_MaxX(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- 1.5
SELECT ST_MaxX(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
-- 3.0
```

6.12.3.22. ST_MaxY

Function declaration:

```
ST_MaxY(geometry)
```

Description: This function returns the maximum Y coordinate of the input geometry.

Example:

```
SELECT ST_MaxY(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- 2.5
SELECT ST_MaxY(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
-- 2.5
```

6.12.3.23. ST_MaxZ

Function declaration:

```
ST_MaxZ(geometry)
```

Description: This function returns the maximum Z coordinate of the input geometry.

Example:

```
SELECT ST_MaxZ(ST_PointZ(1.5, 2.5, 2)) FROM src LIMIT 1;
-- 2
SELECT ST_MaxZ(ST_LineString('linestring z (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
-- 1
```

6.12.3.24. ST_MinX

Function declaration:

```
ST_MinX(geometry)
```

Description: This function returns the minimum X coordinate of the input geometry.

Example:

```
SELECT ST_MinX(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- 1.5
SELECT ST_MinX(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
-- 3.0
```

6.12.3.25. ST_MinY

Function declaration:

```
ST_MinY(geometry)
```

Description: This function returns the minimum Y coordinate of the input geometry.

Example:

```
SELECT ST_MinY(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- 2.5
SELECT ST_MinY(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
-- 2.2
```

6.12.3.26. ST_MinZ

Function declaration:

```
ST_MinZ(geometry)
```

Description: This function returns the minimum Z coordinate of the input geometry.

Example:

```
SELECT ST_MinZ(ST_PointZ(1.5, 2.5, 2)) FROM src LIMIT 1;
-- 2
SELECT ST_MinZ(ST_LineString('linestring z (1.5 2.5 2, 3.0 2.2 1)')) FROM src LIMIT 1;
-- 1
```

6.12.3.27. ST_NumGeometries

Function declaration:

```
ST_NumGeometries(ST_GeometryCollection)
```

Description: This function returns the number of geometries in the input geometry collection.

Example:

```
SELECT ST_NumGeometries(ST_GeomFromText('multipoint ((10 40), (40 30), (20 20), (30 10))'))
FROM src LIMIT 1;
-- 4
SELECT ST_NumGeometries(ST_GeomFromText('multilinestring ((2 4, 10 10), (20 20, 7 8))')) FR
OM src LIMIT 1;
-- 2
```

6.12.3.28. ST_NumInteriorRing

Function declaration:

```
ST_NumInteriorRing(ST_Polygon)
```

Description: This function returns the number of interior rings of the input polygon.

Example:

```
SELECT ST_NumInteriorRing(ST_Polygon(1,1, 1,4, 4,1)) FROM src LIMIT 1;
-- 0
SELECT ST_NumInteriorRing(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1 1))'
)) FROM src LIMIT 1;
-- 1
```

6.12.3.29. ST_NumPoints

Function declaration:

```
ST_NumPoints(geometry)
```

Description: This function returns the number of points in the input geometry.

Example:

```
SELECT ST_NumPoints(ST_Point(1.5, 2.5)) FROM src LIMIT 1;
-- 1
SELECT ST_NumPoints(ST_LineString(1.5,2.5, 3.0,2.2)) FROM src LIMIT 1;
-- 2
SELECT ST_NumPoints(ST_GeomFromText('polygon ((0 0, 10 0, 0 10, 0 0))')) FROM src LIMIT 1;
-- 4
```

6.12.3.30. ST_PointN

Function declaration:

```
ST_PointN(ST_Geometry, n)
```

Description: This function returns the nth 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;
-- POINT(3.0 2.2)
```

6.12.3.31. ST_StartPoint

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;
-- POINT(1.5 2.5)
```

6.12.3.32. ST_EndPoint

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;
-- POINT(3.0 2.0)
```

6.12.3.33. ST_SRID

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
-- Return SRID 0.
```

6.12.4. Operations

6.12.4.1. ST_Aggr_ConvexHull

Function declaration:

```
ST_Aggr_ConvexHull(ST_Geometry)
```


Description: This function returns a convex hull for input geometries by using aggregation transformation.

Example:

```
SELECT ST_Aggr_ConvexHull(geometry) FROM source;
-- Return the convex hull of input geometries from the data source by using aggregation transformation.
```

6.12.4.2. ST_Aggr_Intersection

Function declaration:

```
ST_Aggr_Intersection(ST_Geometry)
```

Description: This function returns the intersection of input geometries by using aggregation transformation.

Example:

```
SELECT ST_Aggr_Intersection(geometry) FROM source;
-- Return the intersection of input geometries from the data source by using aggregation transformation.
```

6.12.4.3. ST_Aggr_Union

Function declaration:

```
ST_Aggr_Union(ST_Geometry)
```

Description: This function returns a union of input geometries by using aggregation transformation.

Example:

```
SELECT ST_Aggr_Union(geometry) FROM source;
-- Return the union of input geometries from the data source by using aggregation transformation.
```

6.12.4.4. ST_Bin

Function declaration:

```
ST_Bin(placeholder)
```

Description: This function returns the bin ID of the input point.

6.12.4.5. 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

Function declaration:

```
ST_Boundary(ST_Geometry)
```

Description: This function returns the boundary of the input geometry.

Example:

```
SELECT ST_Boundary(ST_LineString(0,1, 1,0)) FROM src LIMIT 1;
-- MULTIPOINT((1 0),(0 1))
SELECT ST_Boundary(ST_Polygon(1,1, 4,1, 1,4)) FROM src LIMIT 1;
-- LINESTRING(1 1, 4 1, 1 4, 1 1)
```

6.12.4.7. 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

Function declaration:

```
ST_ConvexHull(ST_Geometry, ST_Geometry, ...)
```

Description: This function returns the convex hull of the input geometry.

Example:

```
SELECT ST_AsText(ST_ConvexHull(ST_Point(0, 0), ST_Point(0, 1), ST_Point(1, 1))) FROM onerow
;
-- MULTIPOLYGON (((0 0, 1 1, 0 1, 0 0)))
```

6.12.4.9. ST_Difference

Function declaration:

```
ST_Difference(ST_Geometry1, ST_Geometry2)
```

Description: This function returns a geometry that indicates the difference between the input geometries.

Example:

```
SELECT ST_AsText(ST_Difference(ST_MultiPoint(1, 1, 1.5, 1.5, 2, 2), ST_Point(1.5, 1.5))) FROM onerow;
-- 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;
-- MULTIPOLYGON (((10 0, 10 10, 0 10, 0 5, 5 5, 5 0, 10 0)))
```

6.12.4.10. ST_Envelope

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.

Example:

```
SELECT ST_Envelope(ST_LineString(0,0, 2,2)) from src LIMIT 1;
-- 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;
-- POLYGON ((2 0, 3 0, 3 3, 2 3, 2 0))
```

6.12.4.11. ST_ExteriorRing

Function declaration:

```
ST_ExteriorRing(polygon)
```

Description: This function returns the exterior ring of a polygon as a linestring.

Example:

```
SELECT ST_ExteriorRing(ST_Polygon(1,1, 1,4, 4,1)) FROM src LIMIT 1;
-- 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;
-- LINESTRING (8 0, 0 8, 0 0, 8 0)
```

6.12.4.12. ST_InteriorRingN

Function declaration:

```
ST_InteriorRingN(ST_Polygon, n)
```

Description: This function returns the nth 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;
-- LINESTRING (1 1, 5 1, 1 5, 1 1)
```

6.12.4.13. ST_Intersection

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.

Example:

```
SELECT ST_AsText(ST_Intersection(ST_Point(1,1), ST_Point(1,1))) FROM onerow;
-- POINT (1 1)
SELECT ST_AsText(ST_Intersection(ST_GeomFromText('linestring(0 2, 0 0, 2 0)'), ST_GeomFromText('linestring(0 3, 0 1, 1 0, 3 0)'))) FROM onerow;
-- 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 onerow;
-- 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 onerow;
-- 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 onerow;
-- MULTIPOLYGON EMPTY or LINESTRING (2 1, 3 1, 2 1)
```

6.12.4.14. ST_SymmetricDiff

Function declaration:

```
ST_SymmetricDiff(ST_Geometry1, ST_Geometry2)
```

Description: This function returns a geometry that consists of the symmetric differences of the input geometries.

Example:

```
SELECT ST_AsText(ST_SymmetricDiff(ST_LineString('linestring(0 2, 2 2)'), ST_LineString('linestring(1 2, 3 2)'))) FROM onerow;
-- 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;
-- 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

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;
-- 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

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

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

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

Function declaration:

```
BOOLEAN ST_EnvIntersects(ST_Geometry1, ST_Geometry2)
```

Description: If the envelopes of geometry1 and geometry2 intersect, this function returns true. Otherwise, this function returns false.

Example:

```
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

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

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

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

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****') fr
om src LIMIT 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*****') fr
om src LIMIT 1;
-- false is returned.
SELECT ST_Relate(st_linestring(0,0, 3,3), ST_linestring(1,1, 4,4), 'T*****') from src LI
MIT 1;
-- true is returned.
SELECT ST_Relate(st_linestring(0,0, 3,3), ST_linestring(1,1, 4,4), '****T****') from src LI
MIT 1;
-- false is returned.
```

6.12.5.9. ST_Touches

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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)'));
+-----+
| _c0    |
+-----+
| 460.6965312526471 |
+-----+
```

6.12.8.6. ST_Area_Sphere

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,
116.259024 40.20199,116.258768 40.201662,116.258376 40.201341,116.258031 40.201036,116.2576
75 40.200734,116.257429 40.200656,116.257357 40.200562,116.257392 40.200051,116.257506 40.1
99433,116.257569 40.198586,116.257564 40.19756,116.257561 40.197372,116.257552 40.197036,11
6.257554 40.19675,116.257539 40.196647,116.257502 40.19653,116.257343 40.196389,116.257153
40.196276,116.256733 40.196071,116.25582 40.195646,116.255628 40.195611,116.255468 40.19565
3,116.255385 40.195742,116.255347 40.195849,116.255258 40.197143,116.255103 40.199576,116.2
55078 40.200585,116.251227 40.20059,116.251098 40.203978,116.259433 40.204111,116.259247 40
.203079,116.259097 40.202114))'));
+-----+
| _c0    |
+-----+
| 353493.765625 |
+-----+
```

6.12.9. R-tree index functions

6.12.9.1. ST_BuildRTreeIndex

Function declaration:

```
RTree ST_BuildRTreeIndex(string uniqueId, string geometryWkt)
```

Description: This function is a user-defined table-valued 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

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

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

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

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

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 ST_Overlaps query.

6.12.9.7. ST_TouchesFromRTree

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

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.

6.12.9.9. ST_KNNFromRTree

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.

6.12.9.10. Example

This topic provides examples on how to use R-tree index functions.

Example 1

```
-- 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));
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;
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 |
+-----+-----+
```

Example 2


```
-- Create an R-tree for all points in a table and use the KNN function to locate 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;
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

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

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, 7432
65 2967450,743265.625 2967416,743238 2967416))', 2249, 4326));
SELECT ST_AsText(ST_TransformWGS84(ST_GeomFromText('POLYGON((-71.1776848522251 42.390289651
2902,-71.1776843766326 42.3903829478009, -71.1775844305465 42.3903826677917,-71.17758259272
31 42.3902893647987,-71.1776848522251 42.3902896512902))')));
```

6.13. SQL Function

User-defined functions (UDFs) in MaxCompute support Java or Python. Some UDFs can be directly implemented by SQL. Therefore, MaxCompute supports SQL functions. This improves the reuse rate of SQL code.

Use SQL functions

Example:

```
FUNCTION ADD(@a BIGINT) AS @a + 1;
SELECT ADD(key), ADD(value) FROM src;
```

Functions as input parameters

Functions can be used as input parameters for SQL functions, including built-in functions, UDFs, and SQL functions.

Example:

```
FUNCTION ADD(@a BIGINT) AS @a + 1;
FUNCTION OP(@a, @fun FUNCTION (BIGINT) RETURNS BIGINT) AS @ fun(@a);
SELECT OP(key, ADD), OP(key, abs) FROM src;
```

Anonymous functions as input parameters

Anonymous functions can be used as input parameters for SQL functions.

Example:

```
FUNCTION OP(@a, @fun FUNCTION (BIGINT) RETURNS BIGINT) AS @ fun(@a);
SELECT OP(key, FUNCTION (@a) AS @a + 1) FROM src;
```

6.14. CLONE TABLE

MaxCompute supports the CLONE TABLE statement. You can execute this statement to clone data from one table to another.

Syntax

```
CLONE TABLE <[src_project_name.]src_table_name> [PARTITION(spec), ...] TO <[dest_project_name.]desc_table_name> [IF EXISTS (OVERWRITE | IGNORE)] ;
```

Note

- If the destination table is not created before data is cloned, a table is created by using the CREATE TABLE LIKE statement when you execute the CLONE TABLE statement.
- If the destination table is created before data is cloned and IF EXISTS OVERWRITE is specified, data in the specified partitions of the destination table is overwritten.
- If the destination table is created before data is cloned and IF EXISTS IGNORE is specified, existing partitions in the destination table are skipped and data in these partitions is not overwritten.

Limits and troubleshooting

- The schema of a destination table must be compatible with that of the source table.
- The CLONE TABLE statement supports both partitioned and non-partitioned tables. Tables that have special data organization structures are not supported. These tables include clustered tables, shard tables, Xlib or Algo tables, and tables with extreme storage.
- Make sure that the configuration of the cluster for the source table intersects with that for the destination table and the data that you want to process is in the same cluster. If any of the conditions is not met, an error is returned.
- If the destination table already exists before data is cloned, you can clone data from a maximum of 10,000 partitions at a time.
- If the destination table does not exist before data is cloned, the number of partitions that you can clone at a time is not limited, which ensures atomicity.
- If a hard link in the Apsara Distributed File System is faulty, purge the recycle bin and try again.
- The user who submits the command must have the Create Table and Update Table permissions on the target project.

Example

The following code shows the partitions and data of the source tables:

```
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.

```
clone table src_copy to src_clone;
odps@ multi>clone table src_copy to src_clone;
ID = 2019102303024544g2540cdv2
OK
odps@ multi>read src_clone;
+-----+-----+
| key    | value  |
+-----+-----+
| 1      | ok     |
+-----+-----+
```

Clone some partitions of the partitioned table.

```
clone table srcpart_copy partition(ds="2008-04-09", hr='11') to srcpart_clone IF EXISTS OVERWRITE;
odps@ multi>clone table srcpart_copy partition(ds="2008-04-09", hr='11') to srcpart_clone IF EXISTS OVERWRITE;
ID = 20191023030534986g4540cdv2
OK
odps@ multi>read srcpart_clone;
+-----+-----+-----+-----+
| key    | value  | ds      | hr      |
+-----+-----+-----+-----+
| 1      | ok49   | 2008-04-09 | 11      |
+-----+-----+-----+-----+
```

Clone data from the partitioned table and skip existing partitions in the destination table.

```
clone table srcpart_copy to srcpart_clone IF EXISTS IGNORE;
odps@ multi>clone table srcpart_copy to srcpart_clone IF EXISTS IGNORE;
ID = 20191023030619196g5540cdv2
OK
odps@ multi>read srcpart_clone;
```

key	value	ds	hr
1	ok49	2008-04-09	11
1	ok48	2008-04-08	12

Clone all data from the partitioned table.

```
clone table srcpart_copy to srcpart_clone2;
odps@ multi>clone table srcpart_copy to srcpart_clone2;
ID = 20191023030825186g6540cdv2
OK
odps@ multi>read srcpart_clone2;
```

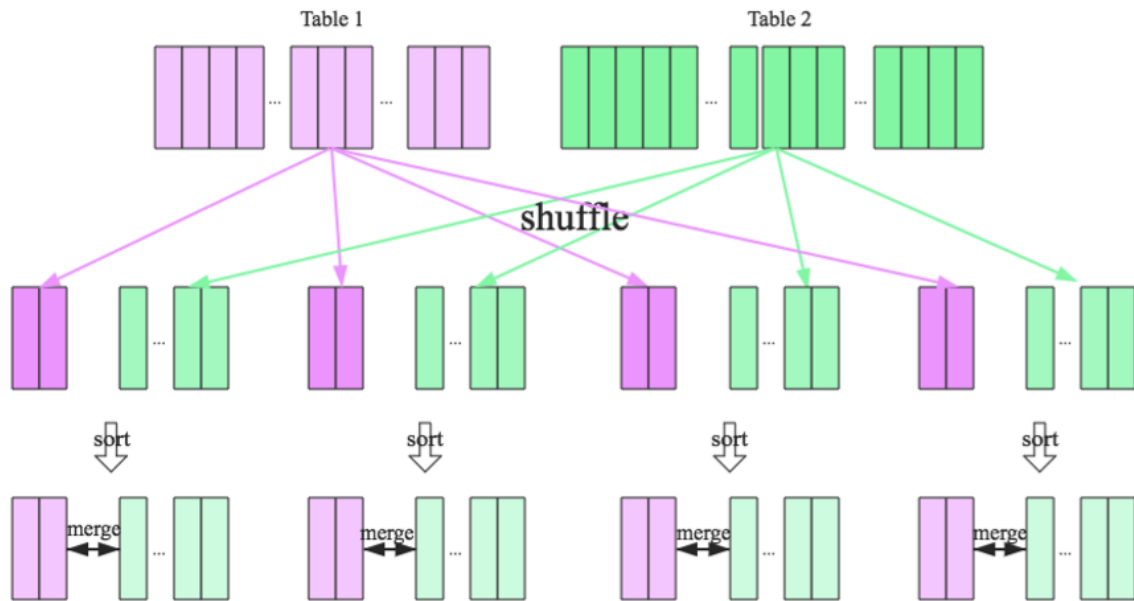
key	value	ds	hr
1	ok49	2008-04-09	11
1	ok48	2008-04-08	12

6.15. MaxCompute Hash Clustering

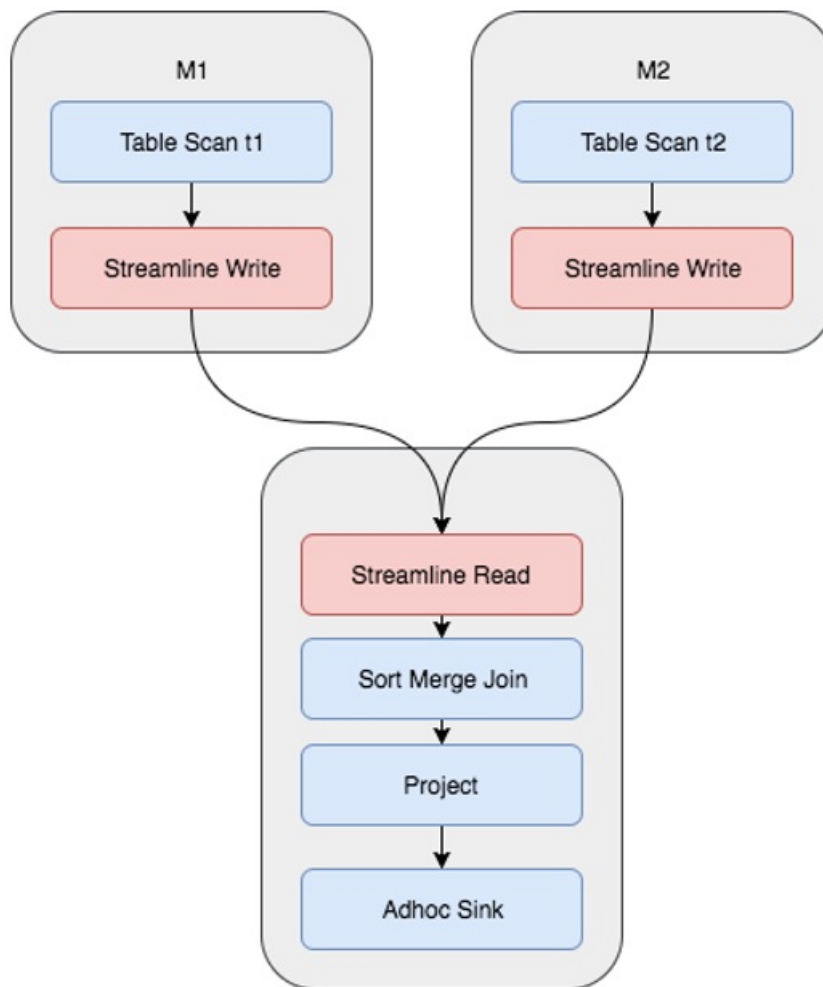
6.15.1. Background information

JOIN operations are commonly used for queries in MaxCompute. MaxCompute provides the following implementation methods of JOIN operations:

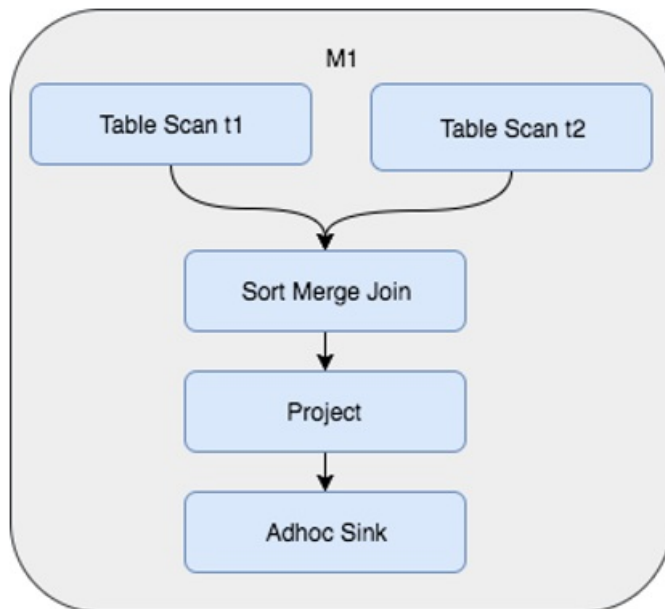
1. **Broadcast hash join:** This method is used when a JOIN operation involves a small table. The small table is broadcasted and transferred to all JoinTask instances. Then, the hash join operation is performed to join the small table with a large table.
2. **Shuffle hash join:** This method is used when a JOIN operation involves large tables that cannot be broadcasted directly. 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 on a JoinTask instance. 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.
3. **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** operation is commonly used in MaxCompute because MaxCompute processes huge volumes of data in most cases. This operation generates repeated shuffle and join operations. The physical execution plan of Job Scheduler of the JOIN operation also requires multiple stages, which consumes excessive volumes of resources.



Therefore, 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 shuffled and sorted repeatedly in subsequent queries. As a result, the number of stages in the physical execution plan of Job Scheduler of a JOIN operation is 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. As a result, MaxCompute optimizes the execution plan, improves the efficiency, and saves resources based on the existing storage characteristics.

6.15.2. Descriptions

6.15.2.1. Enable or disable Hash Clustering

The Hash Clustering feature is available and enabled by default. 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 all the time, contact the MaxCompute team.

Note Clustered indexes improve the efficiency of queries (equivalent values or ranges) based on sort keys. However, you can still experience the superior performance provided by Hash Clustering although you do not add this flag.

6.15.2.2. Create a hash clustering table

You can use the following statement to create a hash clustering 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 too many 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 guarantees that SQL statements can be executed concurrently 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.

6.15.2.3. Modify table attributes

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;
```

Note the following points when you use the ALTER TABLE statement:

- The ALTER TABLE statement can only modify 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 for 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 for the new partitions of a table. Therefore, you cannot specify a partition in this statement.

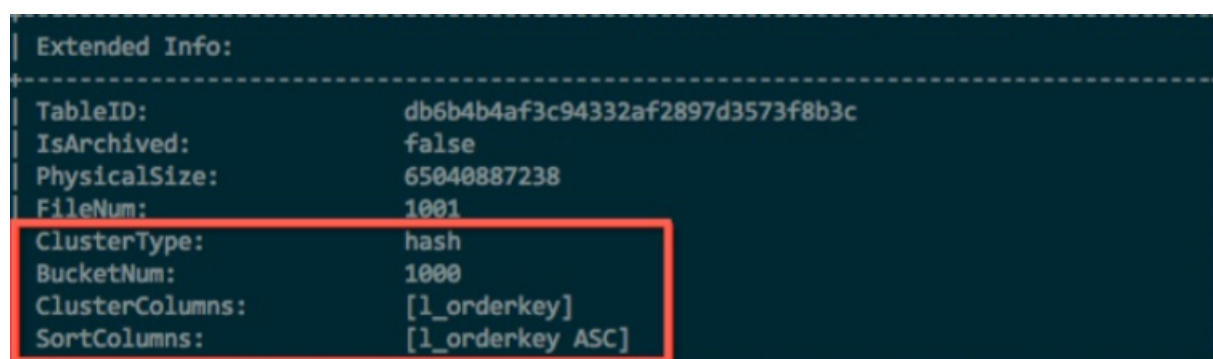
The ALTER TABLE statement is suitable for existing tables. After the Hash Clustering attribute is added, new partitions are stored based on the Hash Clustering attribute.

6.15.2.4. View and verify table attributes

After you create a hash clustering table, execute the following statement to view table attributes:

```
DESC EXTENDED table_name;
```

The Hash Clustering attribute is displayed in Extended Info.



Extended Info:	
TableID:	db6b4b4af3c94332af2897d3573f8b3c
IsArchived:	false
PhysicalSize:	65040887238
FileNum:	1001
ClusterType:	hash
BucketNum:	1000
ClusterColumns:	[l_orderkey]
SortColumns:	[l_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]

6.15.3. Benefits

6.15.3.1. 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. The sample procedure is as follows:

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.

6.15.3.2. Aggregation optimization

The following code provides a syntax sample:

```
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.

6.15.3.3. Storage optimization

In addition to computation optimization, storage space is greatly saved if tables are shuffled and stored in a sorted manner. MaxCompute uses the column store at the underlying layer. Records with the same or similar key-value pairs are stored together by the sort function, which facilitates encoding and compression. As a result, compression efficiency is improved. In some cases, a sorted table can save 50% more storage space than an unsorted table. Therefore, Hash Clustering is suitable for the storage of tables that have long lifecycles.

For example, take a table with 100 GB of TPC-H line items and multiple data types, such as INT, DOUBLE, and STRING. When Hash Clustering is used, about 10% of the storage space is saved while 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_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

6.15.4. ShuffleRemove

- Range clustering tables support the join and aggregate operations. If a join or group key is a range clustering 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 controls whether to enable this feature. This feature is disabled by default.

- If you join two hash clustering tables and the numbers of buckets in these tables are different but are multiples or factors of each other, data redistribution is not required. This improves execution efficiency.

Usage: The `odps.optimizer.enable.hash.partial.repartitioning` flag controls whether to enable this feature. This feature is enabled by default.

- 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.

6.15.5. Limits

The limits of Hash Clustering are described as follows:

- The INSERT INTO statement is not supported. You can only execute the INSERT OVERWRITE statement to import data.
- Small files cannot be merged. Data is evenly distributed in buckets when it is split, so no small files are generated. 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 data uploaded by using Tunnel is unsorted.

In the future, these limits will be resolved. Stay tuned for updates on the official website.

6.16. MaxCompute SQL limits

The following table lists all the limits of MaxCompute SQL statements.

Limits

Item	Maximum value/Limit	Category	Description
Table name length	128 bytes	Length	A table name or column name cannot contain special characters. It can contain only lowercase and uppercase letters, digits, and underscores (_) and must start with a letter.
Comment length	1,024 bytes	Length	A comment can be up to 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.
Statistical definitions of a table	100	Quantity	A table can contain a maximum of 100 statistical definitions.
Statistical definition length of a table	64,000	Length	The length of statistical definitions in a table cannot exceed 64,000.
Screen display	10,000 rows	Quantity	A SELECT statement can generate a maximum of 10,000 rows.
INSERT targets	256	Quantity	A MULTIINS operation can insert a maximum of 256 data tables at a time.
UNION ALL	256 tables	Quantity	A UNION ALL operation can be performed on a maximum of 256 tables.
JOIN sources	128	Quantity	The JOIN operation can be performed on a maximum of 128 source tables.
MAPJOIN memory	512 MB	Quantity	The memory size for all small tables on which the MAPJOIN operation is performed cannot exceed 512 MB.

Item	Maximum value/Limit	Category	Description
Window functions	5	Quantity	A SELECT statement can contain a maximum of five window functions.
PTINSUBQ	1,000 rows	Quantity	A PT IN SUBQUERY statement can generate a maximum of 1,000 rows.
Length of an SQL statement	2 MB	Length	The maximum length of an SQL statement is 2 MB.
Conditions of a WHERE clause	256	Quantity	A WHERE clause can contain a maximum of 256 conditions.
Length of a column record	8 MB	Length	The maximum length of a column record in a table is 8 MB.
IN parameters	1,024	Quantity	This item specifies the maximum number of parameters in an IN clause, such as in(1,2,3,...,1024). Excess parameters can slow down the compilation process. We recommend that you use no more than 1,024 parameters, but this is not a fixed upper limit.
jobconf.json	1 MB	Length	The maximum size of the jobconf.json file is 1 MB. If a table contains a large number of partitions, the size of jobconf.json may exceed 1 MB.
View	Not writable	Operation	A view is not writable and does not support the INSERT operation.
Data type and position of a column	Unmodifiable	Operation	The data type and position of a column are unmodifiable.
Java UDFs	Cannot be abstract or static	Operation	Java UDFs cannot be abstract or static.
Partitions to query	10,000	Quantity	A maximum of 10,000 partitions can be queried.



Notice The preceding MaxCompute SQL limits cannot be modified manually.

6.17. Common MaxCompute SQL parameter settings

6.17.1. MAP configurations

```
set odps.sql.mapper.cpu=100
```

Purpose: It is used to set the number of CPUs for each instance in a Map task. Default value: 100. Value range: 50 to 800.

```
set odps.sql.mapper.memory=1024
```

Purpose: It is used to set the memory size for each instance in a Map task. Default value: 1024 MB. Value range: 256 MB to 12,288 MB.

```
set odps.sql.mapper.merge.limit.size=64
```

Purpose: It is used to set the maximum size of control files to be merged. Default value: 64 MB. You can set this variable to control the inputs of mappers. Value range: 0 to Integer.MAX_VALUE.

```
set odps.sql.mapper.split.size=256
```

Purpose: It is used to set the maximum data input volume for a map. Default value: 256 MB. You can set this variable to control the inputs of mappers. Value range: 1 to Integer.MAX_VALUE.

6.17.2. JOIN configurations

```
set odps.sql.joiner.instances=-1
```

Purpose: It is used to set the number of instances in a JOIN task. Default value: 1. Value range: 0 to 2,000.

```
set odps.sql.joiner.cpu=100
```

Purpose: It is used to set the number of CPUs for each instance in a JOIN task. Default value: 100. Value range: 50 to 800.

```
set odps.sql.joiner.memory=1024
```

Purpose: It is used to set the memory size for each instance in a JOIN task. Default value: 1,024 MB. Value range: 256 MB to 12,288 MB.

6.17.3. Reduce configurations

```
set odps.sql.reducer.instances=-1
```

Purpose: It is used to set the number of instances in a Reduce task. Default value: 1. Value range: 0 to 2,000.

```
set odps.sql.reducer.cpu=100
```


Purpose: It is used to set the number of CPUs for each instance in a Reduce task. Default value: 100.
Value range: 50 to 800.

```
set odps.sql.reducer.memory=1024
```

Purpose: It is used to set the memory size for each instance in a Reduce task. Default value: 1,024 MB.
Value range: 256 to 12,288 MB.

6.17.4. UDF configurations

```
set odps.sql.udf.jvm.memory=1024
```

Purpose: It is used to set the maximum memory size for a UDF JVM heap. Default value: 1,024 MB. Value range: 256 to 12,288 MB.

```
set odps.sql.udf.timeout=600
```

Purpose: It is used to set the timeout value of a UDF. Default value: 600 seconds. Value range: 0 to 3,600 seconds.

```
set odps.sql.udf.python.memory=256
```

Purpose: It is used to set the maximum memory size for UDF python. Default value: 256 MB. Value range: 64 to 3,072 MB.

```
set odps.sql.udf.optimize.reuse=true/false
```

Purpose: after start-up, each UDF function expression can only be calculated once, improving performance. The default is true.

```
set odps.sql.udf.strict.mode=false/true
```

Purpose: It is used to control functions regarding whether to return NULL or error if dirty data is encountered. If it is true, an error is returned. If it is false, NULL is returned.

6.17.5. MAPJOIN configurations

```
set odps.sql.mapjoin.memory.max=512
```

Purpose: It is used to set the maximum memory of a small table in MAPJOIN. Default value: 512 MB. Value range: 128 to 2,048 MB.

```
set odps.sql.reshuffle.dynamicpt=true/false
```

Purpose:

- Some scenarios of dynamic partitioning are time-consuming. Shutting them down can speed up SQL.
- If the dynamic partition value is very small, disabling dynamic partition can avoid data skew.

6.17.6. Configure data skew

```
set odps.sql.groupby.skewindata=true/false
```

Effect: enables the group by optimization.

```
set odps.sql.skewjoin=true/false
```

Effect: enables the join optimization. It takes effect only when odps.sql.skewinfo is configured.

```
set odps.sql.skewinfo
```

Purpose: It is used to set detailed information of join optimization. The command syntax is as follows:

```
set odps.sql.skewinfo=skewed_src:(skewed_key) [ ("skewed_value") ]
```

Example:

The following command is used to set a single skewed data value in a single field:

```
set odps.sql.skewinfo=src_skewjoin1:(key) [ ("0") ]
-- Command output: explain select a.key c1, a.value c2, b.key c3, b.value c4 from src a join
src_skewjoin1 b on a.key = b.key;
```

The following command is used to set multiple skewed data values in a single field:

```
set odps.sql.skewinfo=src_skewjoin1:(key) [ ("0") ("1") ]
-- Command output: explain select a.key c1, a.value c2, b.key c3, b.value c4 from src a join
src_skewjoin1 b on a.key = b.key;
```

6.18. MapReduce-to-SQL conversion for execution

6.18.1. Overview

MaxCompute provides a series of Java APIs for MapReduce to process data.

In the current version, 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.

 Notice

- You do not need to change the original APIs and job logic.
- Only MapReduce jobs of the OpenMR type, which are written with MapReduce APIs, can be converted to SQL.
- This feature can be used 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 clustering tables.
- This feature supports the near-real-time execution of small jobs.

6.18.2. Configure local running settings

1. Download the latest [MaxCompute client](#) package to your computer and properly configure the client.
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 executed by MapReduce. The new compiler, optimizer, and execution engine are not required.

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. When the execution mode is enabled, it affects all jobs. Therefore, the project administrator must apply for and 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 executed by MapReduce. If SQL execution fails in **sql** mode, an error is returned.
- Method 2: Enable the execution mode at the session level. This method is only valid for 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 the job parameters. Example:

```
JobConf job = new JobConf();
job.set("odps.mr.run.mode", "hybrid")
```

The execution mode can be enabled at the project level later by MaxCompute O&M personnel.

6.18.3. Operation settings in DataWorks

Jobs running in DataWorks are updated by the O&M personnel of MaxCompute and DataWorks. You do not need to update the client manually.

1. Enable the conversion for a single job.

You can add the SET statement before a MapReduce job or configure the job parameter for it. These methods take effect at the session level and apply only to the current job.

The following examples demonstrate how to use these methods:

- Add the SET statement, such as `set odps.mr.run.mode=hybrid` .
- Configure the job parameter as follows:

```
JobConf job = new JobConf();  
job.set("odps.mr.run.mode", "hybrid")
```

2. Enable the conversion at the project level by setting *odps.mr.run.mode* for a project.

6.18.4. View running details

You can use Logview and MaxCompute Studio to view MapReduce-to-SQL conversion results and running details of SQL jobs.

1. 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 of the node. Example:

```

create temporary function mr2sql_mapper_152955927079392291755 as 'com.aliyun.odps.map
red.bridge.LotMapperUDTF' using ;
create temporary function mr2sql_reducer_152955927079392291755 as 'com.aliyun.odps.mapr
ed.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_pric
e,v_sku_stock,v_sku_code,v_sku_image,v_delivery_time,v_sku_bulk_order,v_sku_bulk_discou
nt,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_bu
lk_discount,sku_image_version,currency_code ) as (k_id,v_gmt_create,v_gmt_modified,v_pr
oduct_id,v_admin_seq,v_sku_attr,v_sku_price,v_sku_stock,v_sku_code,v_sku_image,v_delive
ry_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_bu
lk_discount,sku_image_version,currency_code ) as (k_id,v_gmt_create,v_gmt_modified,v_pr
oduct_id,v_admin_seq,v_sku_attr,v_sku_price,v_sku_stock,v_sku_code,v_sku_image,v_delive
ry_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_tim
e,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_imag
e,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,curren
cy_code ;

```

2. LogView detail or summary.

You can see that the new execution engine is used to execute jobs.

```

Job run mode: fuxi job
Job run engine: execution engine

```

3. LogView detail or JSON summary.

The JSON summary information in MapReduce only contains the input and output information of 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}]]))
]

```

6.18.5. Perform operations on the distributed file system

Procedure

1. Specify volume files.

You can use either of the following methods to specify volume files:

- o 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.

- o 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.getOutputVolumeFileSystem();

```

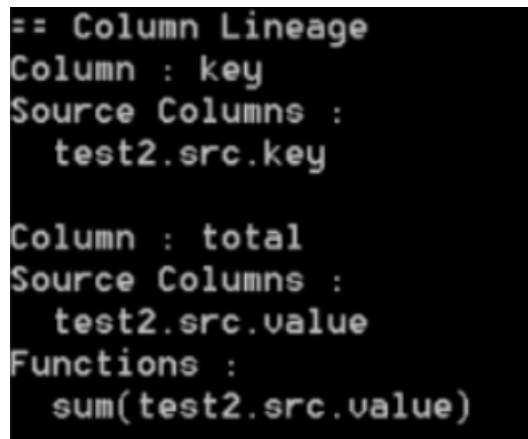
6.19. Analysis of the mapping between SQL input and output fields

6.19.1. Features

MaxCompute SQL provides the feature of analyzing the mapping between SQL input and output fields. This feature is to calculate the fields in the input and output tables based on field mapping. Example:

```
select key, sum(value) as total from src group by key;
```

The following result 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.

6.19.2. Usage notes

This topic describes how to use the feature of analyzing the mapping between input and output fields.

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

Examples

```
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 commit SQL jobs.

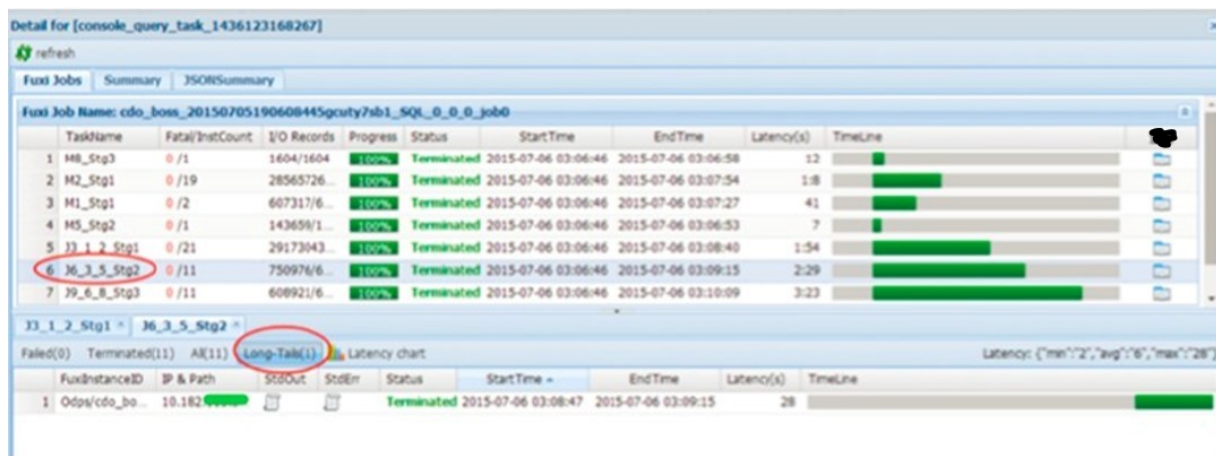
```
odps@ lineage_test>set odps.sql.task.mode=LINEAGE;
OK
odps@ lineage_test>set odps.sql.select.output.format=Humanreadable;
OK
odps@ lineage_test>select * from dual;
== Column Lineage
Column : id
Source Columns :
    test2.dual.id
```

6.20. Common MaxCompute SQL errors and solutions

6.20.1. Data skew

6.20.1.1. Overview

For a running job instance where the min, max, and avg values for the parameters time, input records, and output records are imbalanced (for example, max is much greater than avg), a data skew problem may have occurred. You can check the log view to locate the data skew problem, as shown in the following figure.



The Long Tails tab of each task shows the instance where the data skew occurred. The root cause of data skew is that the amounts of data processed by some instances are much higher than that processed by other instances, causing the running time of these instances to exceed the average time of other instances. As a result, the entire job slows down.

You can reduce the data skew of different SQL data types using different methods.

6.20.1.2. GROUP BY skew

Possible cause: The unbalanced distribution of GROUP BY keys causes data skew in the Reduce step.

Solution: Enable the group skew prevention parameter before running SQL statements:

```
set odps.sql.groupby.skewindata=true
```

Note If this parameter is set to true, the system adds random factors to the shuffle hash algorithm and adds a new task to prevent data skew.

6.20.1.3. DISTRIBUTE BY skew

Possible cause: Using constants for full-table sorting in DISTRIBUTE BY mode will result in data skew at the Reduce end.

Solution: Avoid the preceding operation.

6.20.1.4. JOIN skew

Possible cause: The unbalanced distribution of join on keys (such as a large number of repeated keys in multiple JOIN tables) causes surging Cartesian product data in some JOIN instances, which results in data skew.

Solution: The solutions to different scenarios are as follows:

- If there are small tables on both sides of 'join', perform 'map join' instead of 'join'.
- The skewed key can be dealt with by using individual logic. For example, a large amount of NULL data

in keys on both sides of a table results in skew. In this case, you need to filter out the NULL data before performing the JOIN operation or replacing NULL values with random values by using the CASE WHEN clause, and then do JOIN operation.

- If you do not want to change SQL statements, set 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;
```

6.20.1.5. MULTI-DISTINCT skew

Possible cause: Multiple DISTINCT keywords aggravate the GROUP BY skew problem.

Solution: You can use a two-layer GROUP BY to smooth the skew.

6.20.1.6. Data skew caused by misuse of dynamic partitioning

Possible cause: If dynamic partitioning is enabled, and there are K map instances and N target partitions, a number of small files ($K * N$) may be generated. A large amount of small files can greatly increase the management workload of the file system. Therefore, the following configuration takes effect by default:

```
set odps.sql.reshuffle.dynamiccpt=true;
```

It introduces an additional level of ReduceTask to allow one or more reduce instances to write data to the same target partition. This prevents too many small files from being generated. However, dynamic partition shuffle may cause data skew.

Solution: If there are only a few target partitions, the system will not generate many small files. In this case, you can run the following command to disable the preceding function, or disable dynamic partitioning:

```
set odps.sql.reshuffle.dynamiccpt=false;
```

6.20.2. Quota and resource usage

Computing resources in MaxCompute may be insufficient sometimes because of improper planning and use of cluster resources.

In general, tasks lacking computing resources have two characteristics, one of which is that the task gets stuck with the output remained at a certain stage. For example, in the following figure, the progress of the M1_Stg1 task has stayed at 0% (because R2_1_Stg1 depends on M1_Stg1, it stays at 0% until M1_Stg1 ends).

```

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%]

```

The other characteristic is that the task remains in "Ready" state in the Logview (as shown in the following figure) (a "Ready" task is awaiting allocation of resources; a "Waiting" task is waiting for completion of the dependent task). The "Ready" state indicates that the resources for running these stand-by task instances are insufficient. Once the instances obtain the necessary resources, they resume operating and change to "Running" state.

M1_Stg1					
Failed(0) Ready(5) All(5) Long-Tails(0) Latency chart					
	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 task is split into subtasks based on the execution plan and shown in a DAG, and each subtask invokes multiple instances to execute the computation concurrently. In general, the resources required for invoking an instance are a 1-core CPU and 2 GB of memory. A quota group is assigned to each project for reasonable resource allocation. The quota group determines the maximum amount of resources (CPU and memory) that can be used by all jobs in the project concurrently. Once the resource usage for simultaneously running tasks reaches the limit of the quota group, the tasks are stuck due to insufficient resources.

There are two methods to solve this problem:

- Run the tasks in idle periods.
- Increase the quota group for the project (handled by OAM personnel).

6.20.3. MaxCompute storage optimization tips

Partition tables reasonably

MaxCompute supports the concept of partitioning in a table. A partition refers to the specified partition space in the creation of a table; that is, a few fields in the specified table as the partition columns. In most cases, you can consider a partition as a directory in a file system. MaxCompute divides each value of the partition column into a partition (directory). Users can specify multi-level partitions (use multiple fields of the table as partition columns). Multi-level partitions are like multi-level directories. If you specify the name of the partition that you want to access when using the data, then only the corresponding partition are read, avoiding a full table scan. This improves the processing efficiency and reduces costs.

Example of a partitioning statement: `create table src (key string, value bigint) partitioned by (pt string);` . In this example, `select * from src where pt='20160901';` specifies the partitioning format. MaxCompute takes only the data in the "20160901" partition as the input when generating a query plan.

Example of a non-partitioning statement: `select * from src where key = 'MaxCompute';` scans the entire table.

Partitioning is usually based on date or geographical region. You may also set 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 sale_date is level-1 partition, and region level-2 partition.
```

Set table lifecycle reasonably

Storage space on MaxCompute is precious. You can set the life cycle of a table according to data usage. MaxCompute will delete expired data to save storage space.

Example: Run the `create table test3 (key boolean) partitioned by (pt string, ds string) lifecycle 100;` command to create a table with a lifecycle of 100. If the latest modification time of this table or partition was more than 100 days ago, the table or partition will be deleted.

 **Notice** The lifecycle takes a partition as the smallest unit, so for a partitioned table, if some partitions reach the lifecycle threshold, they will be deleted directly. Partitions that have not reached the lifecycle threshold are not affected.

Run the `alter table table_name set lifecycle days;` command to modify the lifecycle of an existing table.

Archive cold data

Some data need to be preserved either permanently or for a long period of time, but the frequency of access decreases over time. When the use frequency is very low, you can archive the data. The archive function saves data with RAID. Data is not simply stored as three copies. By using the Cauchy Reed-Solomon algorithm, data is stored as six copies of the original data plus three parity blocks. This improves the effective storage ratio from 1:3 to 1:1.5. In addition, MaxCompute uses the bzip2 algorithm to archive tables with a higher compression ratio than other algorithms. Combining the two algorithms reduces storage usage by more than 70%.

Archiving command format is as below:

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

Example:

```
alter table my_log partition(ds='20140101') archive;
```

Merge small files

In the reduce calculation or real-time tunnel data collection, a large number of small files are generated. Too many small files may cause the following problems:

- Many instances are occupied because a single instance can process only a small number of files. This results in a waste of resources, affecting the overall execution performance.
- The file system becomes larger, while the use ratio of disk space becomes smaller.

Currently, there are two alternative ways to merge small files: ALTER merge mode and SQL merge mode:

- The ALTER merge mode merges files through 'console' command. The command format is as follows:

```
ALTER TABLE tablename [PARTITION] MERGE SMALLFILES;
```

- Set control parameters after SQL execution is complete. Run `odps.task.merge.enabled=true;` to determine whether it is necessary to merge small files. If so, start FuxiJob to merge these files.

6.20.4. UDF OOM error

Some jobs will report the OOM error during running. The error message is as follows:

```
FAILED: ODPS-0123144: Fuxi job failed - WorkerRestart errCode:9,errMsg:SigKill(OOM), usually caused by OOM(out of memory)
```

This problem can be solved by setting the UDF runtime parameters:

```
odps.sql.mapper.memory=3072;  
set odps.sql.udf.jvm.memory=2048;  
set odps.sql.udf.python.memory=1536;
```

6.21. Appendix

6.21.1. Escape character

String constants in MaxCompute SQL can be enclosed in single or double quotation marks, in double quotation marks enclosed in single quotation marks, or in single quotation marks enclosed in double quotation marks. Otherwise, they must be expressed with an escape character. Examples of correct expressions: "I'm a happy coder!" and 'I\'m a happy coder!'.

In MaxCompute SQL, the backslash (\) is an escape character, which expresses the special character in a string or interprets the character that follows as the character itself. When a string constant is read, if the backslash is followed by three valid octal digits in the range from 001 to 177, the system converts the ASCII values into the corresponding characters. The following table lists the mappings between escape sequences and represented characters.

Escape sequences

Escape sequence	Represented character
\b	Backspace
\t	Tab
\n	Newline
\r	Carriage return
\'	Single quote
\"	Double quote
\\	Backslash
\;	Semicolon
\Z	Control-Z
\0 or \00	Terminator

Example:

```
select length('a\tb') from dual;  
-- The result is 3, indicating that the string contains three characters, with "\t" regarded as one character. Any character following the escape sequence is interpreted as the character itself.
```

```
select 'a\ab',length('a\ab') from dual;
-- The result is 'aab', with a length of 3. "\a" is interpreted as an ordinary "a".
```

6.21.2. LIKE matching

In LIKE matching, "%" indicates matching any number of characters; "_" indicates matching a single character. If the character "%" or "_" needs to be matched, escape conversion is required. "\\" indicates matching "\", and "_" indicates matching "_".

 **Note** For the character set of strings, MaxCompute SQL currently supports the UTF-8 character set. Data that is encoded in a different format may result in incorrect calculations.

6.21.3. Regular expressions

MaxCompute SQL adopts the PCRE library for regular expressions. Matching is performed character by character. The supported metacharacters are as follows:

- ^: the beginning of a row
- \$: the end of a row
- .: any character
- *: matches zero or multiple times.
- +: matches once or multiple times.
- ?: matches a modifier. If this character follows any one of other delimiters (*, +, ?, {n}, {n,}, or {n,m}), the match is lazy. In the lazy mode, as few strings as possible are matched. In the default greedy mode, as many searched strings as possible are matched zero times or once.
- A|B: A or B
- (abc)*: matches the abc sequence zero or multiple times.
- {n} or {m,n}: the number of matches
- [ab]: matches any character in the brackets.
- [^ab]: ^ represents NOT. This metacharacter matches any character that is neither a nor b.
- \: the escape sequence
- \n: n represents digit 1 to 9. This metacharacter specifies backward reference.
- \d: digit
- \D: non-digit
- [:]: POSIX character set
 - [[:alnum:]]: letter or digit in the range of [a-zA-Z0-9]
 - [[:alpha:]]: letter in the range of [a-zA-Z]
 - [[:ascii:]]: ASCII character in the range of [\x00-\x7F]
 - [[:blank:]]: space and tab in the range of [\t]
 - [[:cntrl:]]: control character in the range of [\x00-\x1F\x7F]
 - [[:digit:]]: digit in the range of [0-9]
 - [[:graph:]]: any character except space in the range of [\x21-\x7E]
 - [[:space:]]: space in the range of [\t\r\n\v\f]
 - [[:print:]]: [[:graph:]] and [[:space:]] in the range of [\x20-\x7E]

- `[[:lower:]]`: lowercase letter in the range of `[a-z]`
- `[[:punct:]]`: punctuation in the range of `[!\"#$%&()*+,-./:;<=>? @^_`{|}~]`
- `[[:upper:]]`: uppercase letter in the range of `[A-Z]`
- `[[:xdigit:]]`: hexadecimal character in the range of `[A-Fa-f0-9]`

The system uses a backslash (`\`) as the escape character, so a backslash (`\`) in a regular expression indicates second escape. For example, the string to be matched by the regular expression is `"a+b"`. The plus sign (`+`) is a special character in regex, and must be escaped to obtain the string `"a+b"`. However, the system needs to escape the first backslash (escape character) before it can be read by regex. Hence, the expression to match `"a+b"` is `"a\\+b"`.

The following example assumes that there is a table named `test_dual`:

```
select 'a+b' rlike 'a\\+b' from test_dual;
+-----+
_c1 |
+-----+
true |
+-----+
```

In extreme cases, to match the character `"\"`, which is a special character in the regular engine, the expression must be `"\"`. The system must perform an escape on the expression, so it is expressed as `"\\\"`.

```
select 'a\\b', 'a\\b' rlike 'a\\\\b' from test_dual;
+-----+-----+
_c0 | _c1 |
+-----+-----+
a\\b | true |
+-----+-----+
```

Note

- If a MaxCompute SQL statement contains `"a\b"`, `'a\b'` is displayed in the output because MaxCompute escapes the expression.
- If a string contains a tab or tab character, the system reads `'\t'` and stores it as one character. Therefore, it is a common character in the regular expression mode.

```
select 'a\tb', 'a\tb' rlike 'a\tb' from test_dual;
+-----+-----+
_c0 | _c1 |
+-----+-----+
a b | true |
+-----+-----+
```

6.21.4. Reserved words

The following are all reserved words in MaxCompute SQL. Do not use these words to name tables, columns, or partitions. Otherwise, an error is returned. Reserved words are case-insensitive.

```
% & && ( ) * +. / ; < <= <> = > >= ? ADD AFTER ALL ALTER ANALYZE AND ARCHIVE ARRAY AS ASC B
EFORE BETWEEN BIGINT BINARY BLOB BOOLEAN BOTH BUCKET BUCKETS BY CASCADE CASE CAST CFILE CHA
NGE CLUSTER CLUSTERED CLUSTERSTATUS COLLECTION COLUMN COLUMNS COMMENT COMPUTE CONCATENATE C
ONTINUE CREATE CROSS CURRENT CURSOR DATA DATABASE DATABASES DATE DATETIME DBPROPERTIES DEFE
RRED DELETE DELIMITED DESC DESCRIBE DIRECTORY DISABLE DISTINCT DISTRIBUTE DOUBLE DROP ELSE
ENABLE END ESCAPED EXCLUSIVE EXISTS EXPLAIN EXPORT EXTENDED EXTERNAL FALSE FETCH FIELDS FIL
EFORMAT FIRST FLOAT FOLLOWING FORMAT FORMATTED FROM FULL FUNCTION FUNCTIONS GRANT GROUP HAV
ING HOLD_DDLTIME IDXPROPERTIES IF IMPORT IN INDEX INDEXES INPATH INPUTDRIVER INPUTFORMAT IN
SERT INT INTERSECT INTO IS ITEMS JOIN KEYS LATERAL LEFT LIFECYCLE LIKE LIMIT LINES LOAD LOC
AL LOCATION LOCK LOCKS LONG MAP MAPJOIN MATERIALIZED MINUS MSCK NOT NO_DROP NULL OF OFFLINE
ON OPTION OR ORDER OUT OUTER OUTPUTDRIVER OUTPUTFORMAT OVER OVERWRITE PARTITION PARTITIONED
PARTITIONPROPERTIES PARTITIONS PERCENT PLUS PRECEDING PRESERVE PROCEDURE PURGE RANGE RCFILE
READ READONLY READS REBUILD RECORDREADER RECORDWRITER REDUCE REGEXP RENAME REPAIR REPLACE R
ESTRICT REVOKE RIGHT RLIKE ROW ROWS SCHEMA SCHEMAS SELECT SEMI SEQUENCEFILE SERDE SERDEPROP
ERTIES SET SHARED SHOW SHOW_DATABASE SMALLINT SORT SORTED SSL STATISTICS STORED STREAMTABLE
STRING STRUCT TABLE TABLES TABLESAMPLE TBLPROPERTIES TEMPORARY TERMINATED TEXTFILE THEN TIM
ESTAMP TINYINT TO TOUCH TRANSFORM TRIGGER TRUE UNARCHIVE UNBOUNDED UNDO UNION UNIONTYPE UNI
QUEJOIN UNLOCK UNSIGNED UPDATE USE USING UTC UTC_TIMESTAMP VIEW WHEN WHERE WHILE
```

6.21.5. New data type settings

If you want to read a table that includes new data types, you are not required to add the `set` `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 returned.
- 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 returned because the implicit data type conversion is invalid.

7. MaxCompute Tunnel

7.1. Overview

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. The data upload and download tools derived from these SDKs meet the requirements of the most common scenarios in which data is migrated to the cloud. The tools also enable you to upload or download data in a variety of other scenarios.

Limits

- Limits on Tunnel-based data uploads:
 - You cannot run Tunnel commands to upload or download data of the ARRAY, MAP, or STRUCT types.
 - 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. Tunnel service connections

DataHub and Tunnel use different endpoints in different network environments. You must also select different endpoints when connecting to the service.

7.3. Selection of cloud data migration tools

MaxCompute provides a variety of data upload and download tools, which can be used in different cloud data migration scenarios. This topic describes the selection of data transmission tools in three typical scenarios.

Hadoop data migration

You can use Sqoop and DataWorks to migrate Hadoop data.

- When you use DataWorks, DataX is required.
- When you use Sqoop, a MapReduce job is executed on the original Hadoop cluster for distributed data transmission to MaxCompute.

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, including 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 for RDS database.

Log collection

You can use tools such as Flume, Fluentd, and Logstash to collect logs.

7.4. 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 the application scenario.

Alibaba Cloud DTplus products

- Data Integration of DataWorks (Tunnel)

Data Integration of DataWorks is a stable, efficient, and scalable data synchronization platform provided by Alibaba Cloud. 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 source types: MaxCompute, ApsaraDB for RDS (MySQL, SQL Server, and PostgreSQL), Oracle, FTP, AnalyticDB (ADS), OSS, ApsaraDB for Memcache, and DRDS.

- MaxCompute client (Tunnel)

Based on the batch data tunnel SDK, the client provides built-in Tunnel commands for data upload and download.

- DTS (Tunnel)

Data Transmission (DTS) is an Alibaba Cloud data service that supports data exchange between multiple data sources, such as Relational Database Management System (RDBMS), NoSQL, and Online Analytical Processing (OLAP) databases. It provides data transmission features, such as data migration, real-time data subscription, and real-time data synchronization.

DTS supports data synchronization from ApsaraDB for RDS and MySQL instances to MaxCompute tables. Other data source types are not supported.

Open-source products

The projects corresponding to each product are open-sourced. You can visit [aliyun-maxcompute-data-collectors](#) to view details.

- Sqoop (Tunnel)

Sqoop 1.4.6 on the community is further developed to provide enhanced MaxCompute support. It can import data from relational databases such as MySQL and data from HDFS or Hive to MaxCompute tables. It can also export data from MaxCompute tables to relational databases such as MySQL.

- Kettle (Tunnel)

Kettle is an open-source ETL tool that is developed in Java. It can run on Windows, Unix, or Linux. It provides graphic interfaces for you to define data transmission topology by using drag-and-drop components.

- Flume (DataHub)

- Apache Flume is a distributed and reliable system. It collects large volumes of log data from different data sources and then aggregates and stores the data in a centralized data storage.
- 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 analysis.
- 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 the Oracle database to DataHub in real time and archive the data in MaxCompute tables.

7.5. Tunnel SDK overview

7.5.1. Overview

Data upload and download tools provided by MaxCompute are compiled based on the Tunnel SDK. This topic describes the major APIs of the Tunnel SDK.

The usage of the SDK varies according to the version. For specific information, see [SDK Java Doc](#).

Major APIs

API	Description
TableTunnel	An entry class of the MaxCompute Tunnel service.
TableTunnel.UploadSession	A session that uploads data to a MaxCompute table.
TableTunnel.DownloadSession	A session that downloads data from a MaxCompute table.
InstanceTunnel	An entry class of the MaxCompute Tunnel service.
InstanceTunnel.DownloadSession	A session that downloads data from a MaxCompute instance. This session applies only to SQL instances that start with the SELECT keyword and are used to query data.

 **Note** The tunnel endpoint supports automatic routing based on the MaxCompute endpoint settings.

7.5.2. TableTunnel

This topic describes the TableTunnel API.

Definition

Definition:

```
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:

- **Lifecycle:** the duration from the creation of the TableTunnel instance to the end of the program.
- TableTunnel provides a method to create UploadSession and DownloadSession objects. TableTunnel.UploadSession is used to upload data, and TableTunnel.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.
- Upload sessions of TableTunnel use the INSERT INTO semantics. Multiple upload sessions of the same table or partition does not affect each other, and the data uploaded in each session is stored in an independent directory.
- In an upload session, each RecordWriter is matched with an HTTP request and is identified by a unique block ID. The block ID is the name of the file corresponding to the RecordWriter.
- If you use the same block ID to enable a RecordWriter multiple times in the same session, the data uploaded by the RecordWriter that calls the close() function last will overwrite all previous data. This feature can be used to retransmit data of a block when data upload fails.

API implementation process

1. The RecordWriter.write() function uploads your data as files to a temporary directory.
2. The RecordWriter.close() function moves the files from the temporary directory to the Data directory.
3. The session.commit() function moves each file in the Data directory to the directory where the corresponding table is located and updates the table metadata. This way, data moved into a table by the current task will be visible to the other MaxCompute tasks such as SQL and MapReduce.

API limits

- The value of a block ID must be greater than or equal to 0 and less than 20000. The size of data to be uploaded in a block cannot exceed 100 GB.
- A session is uniquely identified by its session ID. The lifecycle of a session is 24 hours. If your session times out due to the transfer of large volumes of data, you must transfer your data in multiple sessions.
- The lifecycle of an HTTP request corresponding 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() function, your data may be saved to the buffer and no inbound traffic flows over the corresponding HTTP connection. In this case, you can call the TunnelRecordWriter.flush() function to forcibly flush data from the buffer.

- When you use a RecordWriter to write logs to MaxCompute, the RecordWriter may time out due to unexpected traffic fluctuations. Therefore, we recommend that you:
 - Do not use a RecordWriter for each data record. Otherwise, a large number of small files are generated, because each RecordWriter corresponds to a file. This affects the performance of MaxCompute.
- Do not use a RecordWriter to write data until the size of cached code reaches 64 MB.
- The lifecycle of a RecordReader is 300 seconds.

7.5.3. InstanceTunnel

This topic describes the InstanceTunnel API.

Definition:

```
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);
}
```

Parameter description:

- **projectName**: the name of a project.
- **instanceID**: the ID of an instance.

Limits: Although InstanceTunnel provides an easy way to obtain instance execution results, it is subject to the following permission limits to ensure data security:

- If the number of records does not exceed 10,000, all users who have the read permission on the specified instance can use InstanceTunnel to download the data. This is also applicable to the scenario of calling a Restful API to query data.
- If the number of records exceeds 10,000, only users who have the permission to read all the source tables from which the specified instance queries data can use InstanceTunnel to download the data.

7.5.4. UploadSession

This topic describes the UploadSession interface.

Definition

```
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);
}
```

The following section describes the UploadSession interface.

Description

Item	Description

Item	Description
Lifecycle	The lifecycle of an upload instance starts when the instance is created and ends when data is uploaded.
Create a data upload instance	Create a data upload instance by calling a constructor method or by using TableTunnel. - The synchronous request mode is used. - The server creates a session for the data upload instance and generates a unique upload ID. You can run the getId command to obtain the upload ID.
Upload data	- The asynchronous request mode is used. - Call the openRecordWriter method to generate a RecordWriter. The blockId parameter identifies the data to upload and the position of the data in the table. The value of the blockId parameter ranges from 0 to 20000. If the upload fails, you can upload the data again based on the block ID.
View the status of an upload session	- The synchronous request mode is used. - Call the getStatus method to obtain the status of an upload session. - Call the getBlockList method to obtain the block IDs of successful upload sessions. You must check the block IDs of all upload sessions to identify failed upload sessions. Then, upload the data of failed sessions again.
Complete a data upload	- The synchronous request mode is used. - Call the commit(Long[] blocks) method. The blocks parameter indicates the list of the block IDs of successful upload sessions. The server verifies the list. - The verification enhances data accuracy. If the provided list of block IDs is different from the list on the server, an error is reported.
Status	- UNKNOWN: the initial state of a session. - NORMAL: An upload session is created. - CLOSING: The server sets the upload session to the CLOSING state before it calls the COMPLETE method to complete the data upload. - CLOSED: The data upload is complete. The data is moved to the directory of the result table. - EXPIRED: The upload session times out. - CRITICAL: An error occurs.

 Notice

- 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, you cannot use this block ID to open another RecordWriter.
- The maximum size of a block is 100 GB. Make sure that the volume of data written to each block is greater than 64 MB. Otherwise, computing performance is severely reduced.
- The lifecycle of a session is 24 hours.
- Before you call the openRecordWriter method to write data, we recommend that you prepare data. A network action is triggered every time an openRecordWriter writes 8 KB of data. If no network actions are triggered within 120 consecutive seconds, the server closes the connection and the openRecordWriter becomes unavailable. If this happens, you must open a new openRecordWriter.
- The overwrite mode is added in the COMMIT method. You can use the overwrite mode to submit data. If you use the overwrite mode, the data in this commit overwrites the existing data of a table or partition.

 **Notice** If multiple sessions are concurrently executed and the overwrite mode is used to submit data, undefined behavior is generated. This may cause data inaccuracy. If you use the overwrite mode to submit data, you must control concurrent commits.

7.5.5. DownloadSession

This topic describes the DownloadSession class.

API definition:

```
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 DownloadSession.Status getStatus()
    public RecordReader openRecordReader(long start, long count)
    public RecordReader openRecordReader(long start, long count, boolean compress)
}
```

DownloadSession API description.

DownloadSession API

Parameter	Description
Lifecycle	From the creation of the Download instance to the end of the download process.

Parameter	Description
Purpose	Creates a Download instance by calling a constructor method or using TableTunnel. - Request mode: Synchronous. - The server creates a session for this Download and generates a unique download ID to mark the Download. The console can get data with a get ID. The operation has a high overhead. The server creates indexes for the data files. If many data files exist, the operation takes a long time. Then the server returns the total number of records, and starts concurrent downloads according to the number of records.
Download data	- Request mode: Asynchronous. - Call openRecordReader to generate a RecordReader instance. The Start parameter marks the start position of record for this download. The value of Start is equivalent to or greater than 0. The Count parameter marks the number of records for this download. The value of Count is greater than 0.
View the download process	- Request mode: Synchronous. - Call getStatus to get the download status.
Status	- UNKNOWN: the initial value that is set when the server creates a session. - NORMAL: The download object is successfully created. - CLOSED: The download session is completed. - EXPIRED: The download session times out.

7.5.6. TunnelBufferedWriter

This topic describes the TunnelBufferedWriter interface.

The upload process is complex due to limits on block management and connection timeout on the server. The Tunnel SDK provides an enhanced RecordWriter, TunnelBufferWriter, to simplify the upload process.

The TunnelBufferedWriter interface is defined as follows:

```
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;
}
```

A TunnelBufferedWriter object is described as follows:

- **Lifecycle:** the duration from the time RecordWriter is created to the time the data upload ends.
- **TunnelBufferedWriter instance:** You can call the openBufferedWriter interface of UploadSession

to create a TunnelBufferedWriter instance

- **Data upload:** When you call the Write interface, data is first written to the local cache. After the cache is full, the data is submitted to the server in batches to avoid connection timeout. In addition, if the upload fails, the system automatically retries the upload operation.
- **End upload:** Call the Close interface and then call the Commit interface of UploadSession to end the upload process.
- **Buffer control:** You can use the setBufferSize interface to modify the memory occupied by the buffer (in bytes), preferably 64 MB or more to prevent the server from generating too many small files, which may affect performance. The valid range is 1 MB to 1000 MB. The default value is 64 MB, which is recommended in most cases.
- **Retry policy settings:** You have three retry avoidance policies to choose from: EXPONENTIAL_BACKOFF, LINEAR_BACKOFF, and CONSTANT_BACKOFF. For example, the following code segment sets the Write retry count to 6. To avoid unnecessary retries, each retry is performed only after exponentially ascending intervals of 4s, 8s, 16s, 32s, 64s, and 128s by default.

```
RetryStrategy retry
    = new RetryStrategy(6, 4, RetryStrategy.BackoffStrategy.EXPONENTIAL_BACKOFF)
writer = (TunnelBufferedWriter) uploadSession.openBufferedWriter();
writer.setRetryStrategy(retry);
```

 **Note** We recommend that you do not adjust the preceding settings.

7.6. Tunnel SDK example

7.6.1. Simple upload example

This topic provides a simple upload example of Tunnel SDK.

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,
            table, partitionSpec);
        System.out.println("Session Status is : "
            + uploadSession.getStatus().toString());
        TableSchema schema = uploadSession.getSchema();
        // After data is prepared, run the Writer command to start writing data. The prepared
        data is written to a block.
        // Writing a small volume of data to each block can result in a large number of small
        files.This greatly reduces computing performance. We strongly recommend that you write at l
        east 64 MB (and up to 100 GB) of data to each block.
        // You can estimate the total data volume based on the average data volume and record
        count. For example, 64 MB < Average data volume x Record count < 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 data is written.
            // If no data is transmitted for 120 seconds, the connection times out. The Writer
            command becomes unavailable, and you must write data again.
            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();
    }
}
}
}

```

7.6.2. Simple download example

This topic provides an example for the simple download function of Tunnel SDK.

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.Tu
nnelException;
public class DownloadSample {
    private static String accessId = "<your access id>"; private static String accessKey = "<yo
ur 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 tabl
e 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); od
ps.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, partitionSpe
c);
            System.out.println("Session Status is : "
+ downloadSession.getStatus().toString());
            long count = downloadSession.getRecordCount(); System.out.println("RecordCount is: " + coun
t);
            RecordReader recordReader = downloadSession.openRecordReader(0, count);
            Record record;
            while ((record = recordReader.read()) != null) { consumeRecord(record, downloadSession.get
Schema());
            }
            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();
}
}

```

7.6.3. Multithread upload example

This topic provides a multithread upload example of Tunnel SDK.

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 data is transmitted for 120 seconds, the connection times out. The Writer
                // command becomes unavailable and you must write data again.
                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 the data is prepared, open a writer to start multithread data writing.
            // Writing a small volume of data to each block can result in a large number of small
            // files. This greatly affects computing performance. We recommend that you write at least 64
            // MB (and up to 100 GB) of data to each block.
            // You can estimate the total data volume based on the average data volume and record
            // count. For example, 64 MB < Average data volume x Record count < 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();
        }
    }
}

```

7.6.4. Multithread download example

This topic provides a multithread download example of Tunnel SDK.

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.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.Tu
nnelException;
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 = "<yo
ur 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 tabl
e 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: " + coun
t);
ExecutorService pool = Executors.newFixedThreadPool(threadNum); ArrayList<Callable<Long>> c
allers = 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.shutd
own();
} catch (TunnelException e) { e.printStackTrace();
} catch (IOException e) { e.printStackTrace();
} catch (InterruptedException e) { e.printStackTrace();
} catch (ExecutionException e) { e.printStackTrace();
}
}

```

```
}  
}
```

7.6.5. Example of uploading data by using BufferedWriter

This topic provides an example on how to upload data by using BufferedWriter of the Tunnel SDK.

Example:

```
//Initialize the code of MaxCompute and Tunnel.  
RecordWriter writer = null;  
TableTunnel.UploadSession uploadSession = tunnel.createUploadSession(projectName, tableName  
);  
try {  
    int i = 0;  
    //Generate a TunnelBufferedWriter instance.  
    writer = uploadSession.openBufferedWriter();  
    Record product = uploadSession.newRecord();  
    for (String item : items) {  
        product.setString("name", item);  
        product.setBigint("id", i);  
        //Call the Write interface to write data.  
        writer.write(product);  
        i += 1;  
    }  
} finally {  
    if (writer != null) {  
        //Disable TunnelBufferedWriter.  
        writer.close();  
    }  
}  
//Commit the upload session to end the upload.  
uploadSession.commit();
```

7.6.6. Example of uploading data by using BufferedWriter in multi-threaded mode

This topic provides an example on how to upload data by using BufferedWriter of the Tunnel SDK in multi-threaded 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 Tunnel.
        TableTunnel.UploadSession uploadSession = tunnel.createUploadSession(projectName, tableName);
        UploadThread t1 = new UploadThread(uploadSession);
        UploadThread t2 = new UploadThread(uploadSession);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        uploadSession.commit();
    }
}
```

7.6.7. Examples of uploading and downloading complex data

This topic provides examples of uploading and downloading complex data by using the Tunnel SDK.

Upload complex data

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);
//Import data to a record.
record.setArray(0, arrayData);
record.setMap(1, mapData);
record.setStruct(2, new SimpleStruct((StructTypeInfo) schema.getColumn(2).getTypeInfo(), structData));
//Write the record.
recordWriter.write(record);
```

Download complex data

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);
```

Example of upload and download

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>";
    //Partitions 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 state
ment.
    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": ST
RUCT<name:STRING,age:BIGINT>}.
            UploadSession uploadSession = tunnel.createUploadSession(project, table, partitionSpe
c);
            //Obtain the table schema.
            TableSchema schema = uploadSession.getSchema();
            //Enable the 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);
            //Import data to a record.
            record.setArray(0, arrayData);
            record.setMap(1, mapData);
            record.setStruct(2, new SimpleStruct((StructTypeInfo) schema.getColumn(2).getTypeInfo
(), structData));
            //Write the record.
            recordWriter.write(record);
            //Disable the writer.
            recordWriter.close();
            //Commit the upload session to end the upload.
            uploadSession.commit(new Long[]{0L});
        } catch (TunnelException e) {
            e.printStackTrace();
        }
    }
}

```

```
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();
//Enable the record reader. In the example, one record is read.
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);
System.out.println("download success!") ;
} catch (TunnelException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
```

7.7. Appendix

7.7.1. Tunnel upload/download FAQ

This topic describes frequently asked questions (FAQs) about tunnel upload and download.

What is MaxCompute Tunnel?

Tunnel is data channel of MaxCompute, you are available to upload or download data through Tunnel to or from MaxCompute. You can upload and download only table data (excluding view data) with MaxCompute Tunnel.

Can block IDs be repeated?

Each block ID in an Upload session must be unique. After a block ID is used to start RecordWriter in an upload session, data is written, and the session is closed and committed, this block ID cannot be used to start another RecordWriter. A maximum of 20,000 blocks are supported, with the block IDs ranging from 0 to 19999.

Is there a limit on block size?

The maximum size of a block is 100 GB. We strongly recommend that 64 MB or more data is written into each block. Each block is a file. A file smaller than 64 MB is a small file. Excessive small files will affect the computing performance.

Can a session be shared? Does a session have a life cycle?

Each session has a 24-hour life cycle on the server. It can be used within 24 hours after being created, and can be shared among processes or threads. The block ID of each session must be unique. The procedure for distributed uploading is as follows: **Create a session > Evaluate data volume > Assign blocks (for example, thread 1 uses blocks 0–100 and thread 2 uses blocks 100–200) > Prepare data > Upload data > Commit all blocks with data written.**

How to process write/read timeout or I/O exceptions?

During data uploading, a network action is triggered every time the Writer writes 8 KB data. If no network action is triggered within 120 seconds, the server closes the connection and the Writer becomes unavailable. You have to start a new Writer.

The Reader in data downloading works in a similar way. If no network I/O occurs for a period of time, the connection is closed. We suggest that you run Read without inserting any interfaces from other systems.

Which languages of SDK are available for MaxCompute Tunnel?

MaxCompute Tunnel provides the Java and C++ editions of SDK.

Does MaxCompute Tunnel allow multiple consoles to upload the same table at the same time?

Yes.

Is MaxCompute Tunnel suitable for batch uploading or stream uploading?

MaxCompute Tunnel is more suitable for batch uploading.

Are partitions required for data uploading through MaxCompute Tunnel?

Yes, MaxCompute Tunnel does not automatically build partitions.

What is the relationship between Dship and MaxCompute Tunnel?

Dship is a tool that uploads and downloads data through MaxCompute Tunnel.

Does data uploaded with MaxCompute Tunnel append to the existing file or overwrite the data?

The uploaded data appends to the file.

What is the routing function of MaxCompute Tunnel?

The routing function allows the Tunnel SDK to get the tunnel endpoint by setting MaxCompute. That is, you can run the Tunnel SDK properly by setting the endpoint of MaxCompute.

What is the preferred size of a block when data is uploaded with MaxCompute Tunnel?

The block size depends on factors such as the network situation, real-time performance requirement, data usage, and number of small files in a cluster. If a large volume of data is uploaded continuously, the preferred block size is 64-256 MB. If the data is uploaded in a batch once every day, the block size can be 1 GB.

Why is the timeout error often reported during data downloading with MaxCompute Tunnel?

This may have occurred due to an endpoint error. Check the endpoint configuration. A simple method is to run telnet to check the network connectivity to the endpoint.

Why does the following error occur during data downloading with MaxCompute Tunnel?

```
You have NO privilege 'odps:Select' on {acs:odps:*:projects/XXX/tables/XXX}. project 'XXX' is protected
```

The data protection function has been enabled for the project. Only the project owner has the right to transfer data from one project to another if the project data is protected.

Why does the following error occur during data uploading with MaxCompute Tunnel?

```
ErrorCode=FlowExceeded, ErrorMessage=Your flow quota is exceeded. **
```

The maximum number of concurrent requests is exceeded. By default, MaxCompute Tunnel allows a maximum of 2,000 concurrent upload and download requests (quota). Each request, once it is sent, occupies one quota unit until it ends. Try the following solutions:

- Put the system to sleep, and try again after it awakes.
- Change the concurrency quota to a greater number for the project after obtaining the forecast flow pressure from the administrator.
- Report the problem to the project owner to check and control the top requests occupying a large quota.

7.7.2. Common tunnel error codes

This topic describes common tunnel error codes.

Common tunnel error codes are as follows.

Common error codes

Error code	Error	Processing recommendations
NoSuchPartition	The partition does not exist.	Tunnel doesn't create partitions, you need to create partitions and then upload or download.
InvalidProjectTable	Invalid project name or table name.	Check related names.

Error code	Error	Processing recommendations
NoSuchProject	The project does not exist.	Check related names.
NoSuchTable	The table does not exist.	Check related names.
StatusConflict	The session expires or has been committed.	Re-create the session and upload it.
MalformedDataStream	Data format error.	Normally created because network is disconnected, or Schema and Table are different.
InvalidPartitionSpec	Invalid partition information.	Check partition information. An example of a correct value is pt='1',ct='2017'.
InvalidRowRange	Invalid designated row range, normally it exceeds the max. size or it is 0.	Check related parameters.
Unauthorized	Account information error.	Normally it is wrong AccessId or AccessKey, or local device time has a 15-minute gap with server.
DataStoreError	Storage error.	Contact the administrator.
NoPermission	No permission normally because no related permission or IP whitelist has been set.	Check whether permission is correct.
MissingPartitionSpec	Missing partition information, partition table operation must carry partition information.	Add partition information.
TableModified	Data in the table is modified by other tasks while upload or download.	Re-create the session and re-try.
FlowExceeded	Exceed concurrency quota limit.	Check and control volume of concurrency. If it is needed to add concurrency, please contact the project owner or administrator to evaluate flow pressure.
InvalidResourceSpec	Normally because the project, table or partition information is different from the session.	Check related information and re-try.
MethodNotAllowed	Method is not allowed, normally try to export the view.	Exporting the view is not supported currently.
InvalidColumnSpec	Invalid column information.	Normally it is column name error while download designated column.
DataVersionConflict	It is cross-cluster coping.	Re-try later.

Error code	Error	Processing recommendations
InternalServerError	Internal error.	Re-try or contact the administrator.

8.MaxCompute MapReduce

8.1. Overview

8.1.1. MapReduce

MaxCompute provides a MapReduce programming API. You can use the API to write MapReduce programs to process MaxCompute data.

MapReduce is a distributed data processing model initially proposed by Google. It later gained extensive attention in the industry and was widely used in a variety of business scenarios.

A MapReduce program processes data in two stages: the Map stage and the Reduce stage. It executes the Map stage first and then the Reduce stage. Although you can define the processing logics of Map and Reduce, they need to follow the conventions of the MapReduce framework.

The following is a detailed procedure of how MapReduce processes data:

1. Before you formally start Map, ensure that partition is set for input data. The input data is divided into equal-sized blocks, which are partitions. Each partition is processed as the input of a single Map worker so that multiple Map workers can work together.
2. After partitioning, multiple Map Workers start working simultaneously. Each Map Worker reads its respective shard, computes the shard, and works out the result to Reduce.

 **Note** During data output, each Map worker needs to specify one key for each output data. The key decides the Reduce worker for which the data is targeted. Multiple keys may correspond to a single Reduce worker. Data of the same key is sent to the same Reduce worker, and a single Reduce worker may receive data with different keys.

3. Before entering the Reduce stage, the MapReduce framework will sort the data Key values to make data with the same Key values adjacent. If you specify Combiner, the framework will call Combiner and aggregate data with the same Key.

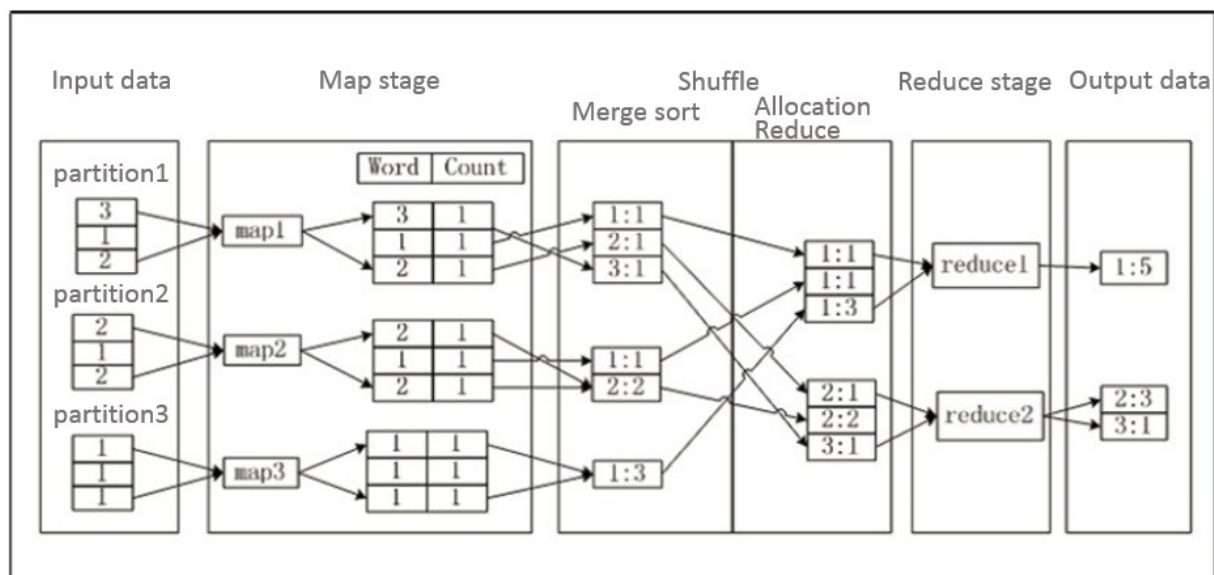
 **Note** You can customize the Combiner logic. Unlike the typical MapReduce framework protocol, MaxCompute requires the input and output parameters be consistent with those of Reduce. This process is generally called Shuffle.

4. When entering the Reduce stage, data with the same Key will be in the same Reduce Worker. A single Reduce Worker may receive data from multiple Map Workers. Each Reduce worker performs the Reduce operation on multiple values with the same key. Finally, the multiple data entries with 1 Key will become 1 Value after the Reduce operation.

 **Note** The preceding section is only a brief introduction to MapReduce. For more details, see related documentation.

The following uses WordCount as an example to explain the related concepts of MaxCompute MapReduce in different stages.

Assume there is a file a.txt, and in each line in the text is a digit. You need to count the number of times each digit appears. Each number is called a Word, and the number of its occurrences is called the Count. To use MaxCompute MapReduce for this purpose, perform the steps shown in the following figure.



1. Partition a.txt data and use data in each partition as input of a single Map worker.
2. For the Map processing input, the Count parameter is set to 1 for each obtained number. The Word|Count pair is output as a Word data key.
3. At the start of the Shuffle stage, the output of each Map worker is sorted by key value (Word value). MapReduce performs the COMBINER operation on the sorted outputs, combining data of the same key (Word value) to form a new Word|Count pair. The process is called combine sorting.
4. Later in the Shuffle stage, data is sent to the Reduce side. The Reduce Worker sorts the received data again depending on Key value.
5. During the Reduce stage, each Reduce Worker uses the same logic as Combiner while processing data, and adds the Count with the same Key value (Word value) to obtain the output result.

Note Because all the MaxCompute data is saved in the table, the input and output of MaxCompute MapReduce can only be a table. Customizing the output format is not permitted, and similar file system APIs are not provided.

8.1.2. Extended MapReduce

In a traditional MapReduce model, data must be stored in a distributed file system (such as an HDFS or a MaxCompute table) after each round of MapReduce operations. A typical MapReduce application is composed of multiple MapReduce jobs. Data is written to a disk after each job is completed. Subsequent map tasks usually only read data once to prepare for the following Shuffle stage. This results in redundant I/O operations.

The computing scheduling logic of MaxCompute supports more complicated programming models. In the preceding case, a reduce operation can be followed by the next reduce operation without having a map operation in between. An extended MapReduce model is provided. This model supports any number of reduce operations after map, such as Map-Reduce-Reduce.

Hadoop Chain Mapper and Chain Reducer also support similar serialized map or reduce operations. However, they are essentially different from the extended MapReduce (MR2) model. Chain Mapper and Chain Reducer are based on the traditional MapReduce model. They support one or more mapper operations, not reducer operations, after the original mapper or reduce operation. One benefit is that you can reuse the preceding mapper business logic by splitting a map or reduce operation into multiple mapper stages. This, however, does not change the underlying scheduling or I/O model.

8.1.3. Open-source compatibility with MapReduce

MaxCompute offers a set of native MapReduce programming models and interfaces. The inputs and outputs for these interfaces are MaxCompute tables, and the data is organized to be processed in the record format.

However, MaxCompute MapReduce APIs differ from Hadoop MapReduce APIs. Traditionally, to migrate your Hadoop MapReduce jobs to MaxCompute, you need to rewrite the MapReduce code, compile and debug the code by using MaxCompute APIs, compress the final code into a JAR package, and upload the package to the MaxCompute platform. This process is tedious and labor-intensive for development and testing. It was expected that the original Hadoop MapReduce code can be used on the MaxCompute platform with little or no modification at all.

To achieve this purpose, the MaxCompute platform provides a plug-in to adapt Hadoop MapReduce to MaxCompute MapReduce. With this plug-in, Hadoop MapReduce jobs are compatible with MaxCompute at the binary level to a certain extent. You can specify some configurations without modifying the code and then run the original Hadoop MapReduce JAR packages on MaxCompute. Click [here](#) to download the plug-in. This plug-in is in the testing stage and does not support custom comparators or key types.

In the following section, a WordCount program is used as an example to introduce the basic usage of this plug-in.

Note

- For more information about open-source compatibility, see [Compatibility with Hadoop MapReduce](#).
- For more information about the Hadoop MapReduce SDK, see the [MapReduce official documentation](#).
- The code in the example is for reference only. You need to modify it based on your business needs.

Download the plug-in

Click [here](#) to download the plug-in. The package name is `openmr_hadoop2openmr-1.0.jar`.

 **Note** This JAR package contains the dependencies with Hadoop 2.7.2. To avoid version conflicts, you must not include Hadoop dependencies in the JAR packages of your jobs.

Prepare a JAR package

Compile and export the WordCount JAR package (`wordcount_test.jar`). The source code of the WordCount program is as follows:

```
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.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. Create input and output tables.

```
create table if not exists wc_in(line string);  
create table if not exists wc_out(key string, cnt bigint);
```

2. Use Tunnel to import data to the input table.

The following content in the data.txt file needs to be imported:

```
hello maxcompute  
hello mapreduce
```

3. You can run the following command on the MaxCompute client to import data from data.txt to wc_in:

```
tunnel upload data.txt wc_in;
```

Configure the mapping between HDFS file paths and MaxCompute tables

The configuration file is named wordcount-table-res.conf.

```

{
  "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 preceding configuration is a JSON file that describes the mapping between HDFS file paths and MaxCompute tables. You need to configure both the input and output. Each HDFS file path matches three configuration items: resolver, tableInfos, and matchMode. The configuration items are described as follows:

- **resolver**: specifies how to process data in files. Currently, the following two built-in resolvers are available: `com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver` and `com.aliyun.odps.mapred.hadoop2openmr.resolver.BinaryFileResolver`. After you specify the resolver name, you must configure properties for the resolver to support data parsing.
 - **TextFileResolver**: regards the input or output as plaintext if the data is of plaintext type. When you configure an input resolver, you must configure the `text.resolver.columns.combine.enable` and `text.resolver.seperator` properties. When `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.seperator`. Otherwise, the first two columns in the input table are used as the key and value fields.

- **BinaryFileResolver**: converts binary data into a data type that is supported by MaxCompute, such as BIGINT, BOOLEAN, and DOUBLE. When you configure an output resolver, you must configure the `binary.resolver.input.key.class` and `binary.resolver.input.value.class` properties. `binary.resolver.input.key.class` defines the key type of the intermediate result, and `binary.resolver.input.value.class` defines the value type.
- **tblInfos**: specifies the MaxCompute table that corresponds to HDFS. At present, only the `tblName` parameter is configurable. The `partSpec` and `label` parameters must be set to the values the same as those in the preceding example.
- **matchMode**: specifies the path matching mode. It can be set to `exact` or `fuzzy`. You can use a regular expression in `fuzzy` mode to match the HDFS input path.

Submit a job

Use the MaxCompute command line tool `odpscmd` to submit the job. Run the following command in `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 mapped configuration file configured with `/foo/bar`.
- **wordcount_test.jar**: the JAR package of your Hadoop MapReduce program.
- **com.aliyun.odps.mapred.example.hadoop.WordCount**: the class name of the job that you want to run.
- **/foo/bar**: the path on HDFS, which is mapped to `wc_in` and `wc_out` in the JSON configuration file.
- After you configure the mapping, you must import the Hadoop HDFS input file to `wc_in` for MapReduce computing by using the data integration function of DataX or DataWorks, and export the result table `wc_out` to your HDFS output directory `/bar`.
- Before you run the preceding command, make sure that `hadoop2openmr-1.0.jar`, `wordcount_test.jar`, and `wordcount-table-res.conf` have been stored in the current directory of `odpscmd`. Otherwise, you must make the relevant changes when you specify the configuration and `-classpath`.

The following figure shows the running process.

```
odps@ zhe-jar -D0095,000096,1861,303,CONF=/wordcount-table-foo.conf -classpath hadoopqpsmr-1.0-jar,wordcount_test.jar com.aliyun.odps.mapred.example.hadoop.WordCount /foo /bar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#staticLoggerBinder for further details.
Running job in console.
[INFO] deprecation - mapred.job.map.memory.mb is deprecated. Instead, use mapreduce.map.memory.mb
[INFO] deprecation - mapred.job.reduce.memory.mb is deprecated. Instead, use mapreduce.reduce.memory.mb
http://logview.odps.aliyun-inc.com:8080/login/?w=http://72.181.234.140:8100/api/?w=http://20100912005518095glmfekstoken=h22u8021c7d1wntT300Wu15207b782718P5uP9F8T300CTauwY7070M
Uqwa3c1LC3520Wu8Uj25180y307306b20wz0p0y6p1Y30u13p0250p0w0M5J20W4J0u0J0M73u00U3Mg107Yv0u810Wv0E119K5w1W0yc21v01302j01RQ==
InstanceId: 20100912005518095glmfekstoken=h22u8021c7d1wntT300Wu15207b782718P5uP9F8T300CTauwY7070M
[INFO] Job - The url to track the job: http://logview.odps.aliyun-inc.com:8080/login/?w=http://72.181.234.140:8100/api/?w=http://20100912005518095glmfekstoken=h22u8021c7d1wntT300Wu15207b782718P5uP9F8T300CTauwY7070M
Uqwa3c1LC3520Wu8Uj25180y307306b20wz0p0y6p1Y30u13p0250p0w0M5J20W4J0u0J0M73u00U3Mg107Yv0u810Wv0E119K5w1W0yc21v01302j01RQ==
...
2016-09-12 16:55:33 MR_Job0-0/0/1[OK]  MR_1_Job0-0/0/1[OK]
2016-09-12 16:55:41 MR_Job0-0/1/1[100%] MR_1_Job0-0/0/1[OK]
...
Inputs:
  zhe_wc_in: 2 (400 bytes)
Outputs:
  zhe_wc_out: 3 (376 bytes)
MR_Job_20100912005518095glmfekstoken=h22u8021c7d1wntT300Wu15207b782718P5uP9F8T300CTauwY7070M
  Worker Count:1
  Input Records:
    Input: 2 (min: 2, max: 2, avg: 2)
  Output Records:
    MR_1: 3 (min: 3, max: 3, avg: 3)
MR_1_Job_20100912005518095glmfekstoken=h22u8021c7d1wntT300Wu15207b782718P5uP9F8T300CTauwY7070M
  Worker Count:1
  Input Records:
    Input: 3 (min: 3, max: 3, avg: 3)
  Output Records:
    MR_1FS_DataLink_0: 3 (min: 3, max: 3, avg: 3)

Counters: 0
OK
odps@ zhe-
```

After the job running process is complete, you can view the result table wc_out to check whether the job is successful and whether the results meet expectations.

```
odps@ zhe>read wc_out;
+-----+-----+
| key      | cnt |
+-----+-----+
| hello    | 2   |
| mapreduce | 1   |
| maxcompute | 1   |
+-----+-----+
odps@ zhe-
```

8.2. Features

8.2.1. Run command

The MaxCompute client provides a jar command for running MapReduce jobs.

Command 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
```

The following table describes the parameters.

Parameters

Parameter	Description
-conf <configuration file>	Indicates a JobConf file.
-classpath <local_file_list>	Indicates the classpath for local execution. It specifies the local paths (including relative path and absolute path) of the jar packet where the main function is located.
-D <prop_name>=<prop_value>	Indicates the java attribute of <mainClass> in local execution. You can define multiple attributes.
-local	Indicates that the MapReduce job is run locally. It is mainly used for program debugging.
-resources <resource_name_list>	Declares the resources used by a MapReduce job. Typically, you must specify the name of the resource where the Map or Reduce function is located in resource_name_list.  Notice If the Map or Reduce function reads from other MaxCompute resources, you must add the names of these resource to resource_name_list. Multiple resources are separated by commas. If you want to use resources in another project, you must append PROJECT_NAME/resources/ before the resource name, such as -resources otherproject/resources/resfile.

 **Note** The preceding optional parameters are included in <GENERIC_OPTIONS>.

You can use the -conf option to specify the JobConf file. This file can affect the settings of JobConf in the SDK. For more information about JobConf, see the introduction of MapReduce core interfaces. The following is an example of a JobConf file.

Example:

```
<configuration>
<property>
<name>import.filename</name>
<value>resource.txt</value>
</property>
</configuration>
```

In the preceding example, the JobConf file is used to define a variable named import.filename. The value of this variable is resource.txt. You can use the JobConf API in MapReduce to obtain the value of this variable. You can also use the JobConf API 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
```

8.2.2. Concepts

8.2.2.1. MapReduce

Map and Reduce functions support setup and cleanup methods for their corresponding map() and reduce() methods. A setup method is called prior to the Map() or Reduce() method. Each worker calls it once. A cleanup method is called after the Map() or Reduce() method. Each worker calls it once.

 **Note** For more information about the usage example, see [Example program](#).

8.2.2.2. Sorting

Columns in the key records output by map can be set as sort columns. Custom comparators are not supported. You can select a few sort columns as group columns. Custom group comparators are not supported. Sort columns are generally used to sort user data, while group columns are used for secondary sorting.

 **Note** For detailed examples, see [Secondary sorting source code](#).

8.2.2.3. Partition

MaxCompute supports partition columns and custom partitioners. Partition columns take precedence over custom partitioners. Partitioners are used to allocate Map output data to different Reduce workers based on the partition logic.

8.2.2.4. Combiner

The Combiner function combines adjacent records in the Shuffle stage. You can choose to use the Combiner function based on your business logic. The Combiner function is an optimized MapReduce computing framework. The Combiner logic is the same as the Reduce logic. After map outputs data, the framework merges data with the same key value on the map side locally.

8.2.2.5. Submit a job

This topic describes how to use the MaxCompute client to run and submit a MapReduce job.

The MaxCompute client provides a JAR command for running MapReduce jobs. 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,
seperate by comma
    -classpath <local_file_list>         classpaths used to run mainClass
    -D <name>=<value>                    Property value pair, which will be used to run m
ainClass
    -l                                   Run job in local mode

```

Example:

```

jar -conf \home\admin\myconf -resources a.txt,example.jar -classpath ..\lib\example.jar:.\o
ther_lib.jar -D java.library.path=.\native;

```

<GENERIC_OPTIONS> includes the following options, which are all optional:

- **-conf <configuration file>**: specifies the JobConf file. This file can affect the settings of JobConf in the SDK.
- **-resources <resource_name_list>**: declares the resources used for running the 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 from other MaxCompute resources, you must add the names of these resources to resource_name_list. Separate multiple resources with commas (,). If you want to use resources in another project, you must append PROJECT/resources/ to the beginning of the resource name, such as -resources otherproject/resources/resfile.

- **-classpath <local_file_list>**: specifies the classpath for local execution. It is used to specify the local paths (including relative and absolute paths) of the JAR package where the main function is located. Package names are separated with default file delimiters of the system. In most cases, semicolons (;) are used in Windows, and commas (,) are used in Linux. 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 written in the same package. In this case, the -resources and -classpath options both contain information such as mapreduce-examples.jar. However, -resources references the Map or Reduce function and is executed in a distributed environment, while -classpath references the main function and is executed locally with the specified JAR package saved in a local directory.

- **-D <name>=<value>**: specifies the Java attribute of <mainClass> during local execution. You can define multiple attributes.
- **-l** specifies that the MapReduce job is run locally. It is mainly used for program debugging.

The following code is an example of the JobConf file:

```

<configuration>
  <property>
    <name>import.filename</name>
    <value>resource.txt</value>
  </property>
</configuration>

```

In the preceding example, the JobConf file is used to define a variable named `import.filename`. The value of this variable is `resource.txt`. You can use the JobConf API in MapReduce to obtain the value of this variable. You can also use the JobConf API in the SDK for the same purpose.

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;
```

8.2.2.6. Input and output

- MaxCompute MapReduce supports the following build-in data types: BigInt, Double, String, Datetime, Boolean, Decimal, Tinyint, Smallint, Int, Float, Varchar, Timestamp, Binary, Array, Map, and Struct. MapReduce does not support custom data types.
- MapReduce supports input from multiple tables with different schemas. You can use the `map` function to obtain the table information corresponding to the current record.
- MapReduce supports NULL as input, but does not support views as input.
- Reduce can write output to different tables or different partitions of a table. The target tables can have different schemas. Each output is identified by a label. An output is not labeled by default. At least one output is generated.

 **Note** For examples, see [Example programs](#).

8.2.2.7. Read data from resources

You can use the `Map` or `Reduce` function to read data from MaxCompute resources. Any `Map` or `Reduce` worker loads resources to the memory for you to write code.

 **Note** For a detailed example, see [Resource utilization example](#).

8.2.2.8. Run MapReduce tasks locally

You can specify the `-local` parameter in the `jar` command to enable local debugging by simulating the running of a local MapReduce instance. During local running, the client downloads from MaxCompute the metadata and data of the input table, required resources, and the metadata of the output table needed for local debugging. The client saves the data to a local directory named `warehouse`. When MapReduce running is complete, MapReduce saves the computing results to a file in the `warehouse` directory. If the input table and required resources are already downloaded to the local `warehouse` directory, MapReduce directly references the data and files in the `warehouse` directory the next time it runs, instead of downloading the data again.

A MapReduce instance that is running locally may start multiple map or reduce processes, which run serially rather than concurrently. The simulated running process is different from an actual distributed running process in the following aspects:

- Input table row count: Up to 100 rows of data can be downloaded.
- Resource usage: In a distributed environment, MaxCompute limits the size of referenced resources. For more information, see [Application limits](#). During local running, the size of resources is unlimited.
- Security limits: MaxCompute MapReduce and UDF programs running in a distributed environment are subject to Java sandbox restrictions. There is no such restrictions during local running.

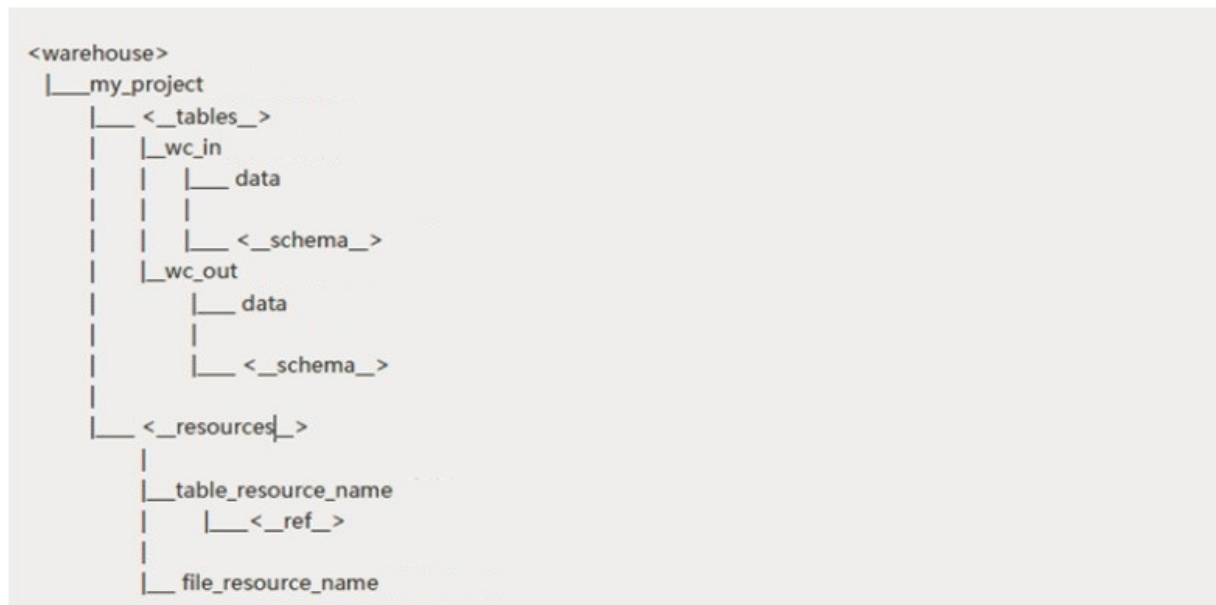
The following is a simple local running example.

Example:

```
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 a WordCount example, see [WordCount example](#).

If you run the local debugging command for the first time, a directory named warehouse is created in the current path after the command is complete. The following figure shows the directory structure of warehouse.



Directories of the same level as my_project are projects. wc_in and wc_out are data tables. The table file data that you read or write using the jar command is downloaded to directories of this level. The schema file contains the metadata of a table in the following format:

```

project=local_project_name
table=local_table_name
columns=col1_name:col1_type,col2_name:col2_type
partitions=p1:STRING,p2:BIGINT
-- This field is not required in this example.

```

Note Separate the name and type of a column with a colon (:), and separate columns with commas (,). You need to declare the project name and table name at the top of the schema files as projectname.tablename. Separate the declaration from column definition with a comma (,). The data file stores table data. The number of columns and data values must match the definition in the schema file. Separate columns with commas (,).

Example of wc_in schema file:

```
my_project.wc_in,key:STRING,value:STRING
```

Example of the corresponding data file:

```
0,2
```


 **Note** 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, the client directly uses the data in the wc_in directory, instead of downloading it again. Note that you can download data from MaxCompute only when running MapReduce locally. If you use the Eclipse plugin for local debugging, you cannot download data from MaxCompute.

Example of wc_out schema file:

```
my_project.wc_out, key:STRING, cnt:BIGINT
```

Example of the corresponding data file:

```
0,1  
2,1
```

 **Note** The client downloads the metadata of wc_out from MaxCompute, and saves the data to the schema file. The data file is generated to store the local running results. You can also compile schema and data files, and save them in the corresponding table directory. Then, when you run MapReduce locally, the client detects these files in the table directory, and will not download data from MaxCompute. The local table directory does not have to correspond to an actual table in MaxCompute.

The following table compares the features of Hadoop MapReduce with MaxCompute MapReduce.

Feature comparison

Feature	Hadoop MapReduce	MaxCompute MapReduce
Task progress report	The map stage calculates the read data volume. The Reduce step monitors the progress of various phases. Based on this information, the overall task progress can be estimated. The client can obtain real-time task progress. Task progress is not controlled by users.	Real-time task progress reporting is supported in a different way.
Statistics	Custom statistics, real-time summary, and real-time updates are supported. Real-time statistics updates are not controlled by users.	Real-time statistics updates are not supported during runtime. Calculation and summary are performed after a task is completed.
File compression	Users have an option to specify compressed storage.	File compression is not supported. Users cannot directly store files.
Speculative execution	/	Speculative execution is enabled by default, and cannot be configured by users.

Feature	Hadoop MapReduce	MaxCompute MapReduce
End of task notification	The server notifies the client of task completion.	No notifications are provided. When using the SDK to submit tasks, users must poll task status constantly until the tasks are completed.

8.3. SDK introduction

8.3.1. Major API overview

Major APIs

API	Description
MapperBase	User-defined Map functions must inherit this class. It converts the record objects in the input table into key-value pairs, and outputs them to the Reduce stage or directly to the result table by skipping the Reduce stage. The jobs that skip the Reduce stage and directly output calculation results are also called Map-Only jobs.
ReducerBase	User-defined Reduce functions must inherit this class. It reduces a set of values associated with a key.
TaskContext	One of the input parameters of multiple member functions of MapperBase and ReducerBase. It contains the contextual information of task execution.
JobClient	It is used to submit and manage jobs, including the blocking mode (synchronous) and non-blocking mode (asynchronous).
RunningJob	Job runtime objects for tracking running MapReduce job instances.
JobConf	Describes the configuration of a MapReduce task. The JobConf object is generally defined in the main program (main function). Then, JobClient submits the job to MaxCompute.

8.3.2. API description

8.3.2.1. MapperBase

The following table lists the major function APIs.

Major APIs

API	Description
<code>void cleanup(TaskContext context)</code>	The Map method is called after the map stage ends.
<code>void map(long key, Record record, TaskContext context)</code>	Map method for processing records in the input table.
<code>void setup(TaskContext context)</code>	The Map method is called before the map stage begins.

8.3.2.2. ReducerBase

The following table lists the major function APIs.

Major APIs

API	Description
<code>void cleanup(TaskContext context)</code>	The Reduce method is called after the reduce stage ends.
<code>void reduce(Record key, Iterator<Record > values, TaskContext context)</code>	Reduce method, processing records in input table.
<code>void setup(TaskContext context)</code>	The Reduce method is called before the reduce stage begins.

8.3.2.3. TaskContext

The following table lists the major function APIs.

Major APIs

API	Description
<code>TableInfo[] getOutputTableInfo()</code>	Get output table information.
<code>Record createOutputRecord()</code>	Create record object of default output table.
<code>Record createOutputRecord(String label)</code>	Create record object of given label output table.
<code>Record createMapOutputKeyRecord()</code>	Create record object of Map output Key.
<code>Record createMapOutputValueRecord()</code>	Create record object of Map output Value.
<code>void write(Record record)</code>	Write record to default output for writing data at Reduce end. Can be called multiple times at Reduce end.
<code>void write(Record record, String label)</code>	Write record to given label output for writing data at Reduce end. Can be called multiple times at Reduce end.
<code>void write(Record key, Record value)</code>	Write record to intermediate result for writing data at Map end. Can be called multiple times at Map end.
<code>BufferedInputStream readResourceFileAsStream(String resourceName)</code>	Read file type resource.
<code>Iterator<Record > readResourceTable(String resourceName)</code>	Reads table type resources.

API	Description
Counter <code>getCounter(Enum<? > name)</code>	Get Counter object of given name.
Counter <code>getCounter(String group, String name)</code>	Get Counter object of given group name and name.
void <code>progress()</code>	Report heartbeat information to the MapReduce framework. If the processing time of your method is extended, and you have not called the framework during the current process, call it to avoid a task timeout period. 600 seconds is set as the timeout period by default.

 **Note** TaskContext API has a progress function which prevents it from being forced out due to time-out if a worker runs for a long time. This API sends heartbeats to the framework rather than reporting the Worker progress. The default worker timeout period for MaxCompute MapReduce is 10 minutes (which cannot be modified by the user). If the worker does not send a heartbeat (calling the progress API) in 10 minutes, the framework terminates the worker and the MapReduce task fails. We recommend that you let the worker call the progress API periodically in the Mapper/Reducer function to prevent the framework from terminating the task unexpectedly.

8.3.2.4. JobConf

The following table lists the major function APIs.

Major APIs

API	Description
void <code>setResources(String resourceNames)</code>	Declare resources this job uses. Only declared resources are available to be read through TaskContext object while run Mapper/Reducer.
void <code>setMapOutputKeySchema(Column[] schema)</code>	Set attribute of Key output from Mapper to Reducer.
void <code>setMapOutputValueSchema(Column[] schema)</code>	Set attribute of Value output from Mapper to Reducer.
void <code>setOutputKeySortColumns(String[] cols)</code>	Set sorting columns of Key output from Mapper to Reducer.
void <code>setOutputGroupingColumns(String[] cols)</code>	Set Key grouping columns.
void <code>setMapperClass(Class<? extends Mapper> > > theClass)</code>	Set Mapper function of job.
void <code>setPartitionColumns(String[] cols)</code>	Set partition columns designated by job. It is all the columns of Mapper output Key by default.

API	Description
void setReducerClass(Class<? extends Reducer> theClass)	Set the job reducer.
void setCombinerClass(Class<? extends Reducer> theClass)	Set job combiner. When run at Map side, the function is similar to a single Map and local Key value of the local Reduce.
void setSplitSize(long size)	Set input split size, unit: MB, default value: 640.
void setNumReduceTasks(int n)	Set number of Reducer task, it is 1/4 number of Mapper task by default.
void setMemoryForMapTask(int mem)	Set memory size of single Worker in Mapper tasks, unit: MB, default value: 2048.
void setMemoryForReduceTask(int mem)	Set memory size of single Worker in Reduce tasks, unit: MB, default value: 2048.
void setOutputSchema(Column[] schema, String label)	Set output attribute of designated label. While multiplexed output, each output corresponds to 1 label.

Note

- Normally, GroupingColumns is included in KeySortColumns, and KeySortColumns is included in Key.
- At the Map side, the Record output from Mapper calculates Hash value and then, based on the set PartitionColumns, determines which Reducer to distribute to for sorting Records based on KeySortColumns.
- At the Reduce side, after sorting input Records based on KeySortColumns, Records are grouped based on columns designated by GroupingColumns, and are input in traversal sequence. The use of the same Records in columns designated by GroupingColumns as 1 input called by reduce function.

8.3.2.5. JobClient

The following table lists the major function APIs.

Major APIs

API	Description
static RunningJob runJob(JobConf job)	Use blocking (synchronous) mode to submit MapReduce jobs and return immediately.
static RunningJob submitJob(JobConf job)	Use non-blocking (asynchronous) mode to submit MapReduce jobs and return immediately.

8.3.2.6. RunningJob

The following table lists the major function APIs.

Major APIs

API	Description
String <code>getInstanceID()</code>	Get instance ID for checking run log and job management.
boolean <code>isComplete()</code>	Check whether job is complete.
boolean <code>isSuccessful()</code>	Check whether job instance is successful.
void <code>waitForCompletion()</code>	Wait until job instance is complete. It is typically used for jobs submitted in asynchronous mode.
JobStatus <code>getJobStatus()</code>	Check job instance status.
void <code>killJob()</code>	End the job.
Counters <code>getCounters()</code>	Get Counter information.

8.3.2.7. InputUtils

The following table lists the major function APIs.

Major APIs

API	Description
static void <code>addTable(TableInfo table, JobConf conf)</code>	Add table to task input. It can be called more than once. Newly added tables are appended to input queue.
static void <code>setTables(TableInfo [] tables, JobConf conf)</code>	Add tables to task input.

8.3.2.8. OutputUtils

The following table lists the major function APIs.

Major APIs

API	Description

API	Description
<code>static void addTable(TableInfo table, JobConf conf)</code>	Adds a table to the task output. It can be called more than once. Newly added tables are appended to the output queue.
<code>static void setTables(TableInfo [] tables, JobConf conf)</code>	Adds multiple tables to the task output.

8.3.2.9. Pipeline

Pipeline is the main class of MR2. A Pipeline can be built with Pipeline.Builder. Major Pipeline APIs are as follows:

```
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)
```

Example:

```
job job = new job ();
pipeline pipeline = pipeline. builder ()
    . Addmapper (maid. 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 build MapReduce tasks of a Map operation followed by two Reduce operations in the MAIN function. If you are familiar with the basic features of MapReduce, you can use MR2 easily. We also recommend that you learn the basic features of MapReduce before using MR2, namely configuring MapReduce tasks through JobConf. JobConf can only configure MapReduce tasks of a Map operation followed by a single Reduce operation.

8.3.3. Compatibility with 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
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

Type	Interface	Compatible with Hadoop MapReduce?
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
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

Type	Interface	Compatible with Hadoop MapReduce?
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(TaskAttempt ID taskId)	No. An exception is reported.
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.

Type	Interface	Compatible with Hadoop MapReduce?
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
Job	setMapOutputValueClass(Class<? > theClass)	Yes

Type	Interface	Compatible with Hadoop MapReduce?
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

Type	Interface	Compatible with Hadoop MapReduce?
Job	setTaskOutputFilter(Configuration conf, org.apache.hadoop.mapreduce.Job.TaskStatusFilter newValue)	No. An exception is reported.
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

Type	Interface	Compatible with Hadoop MapReduce?
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
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.

Type	Interface	Compatible with Hadoop MapReduce?
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.
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.

Type	Interface	Compatible with Hadoop MapReduce?
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.
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

Type	Interface	Compatible with Hadoop MapReduce?
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
JobConf	setNumReduceTasks(int n)	Yes
JobConf	setNumTasksToExecutePerJvm(int numTasks)	Empty implementation
JobConf	setOutputCommitter(Class<? extends OutputCommitter> theClass)	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce?
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
JobConf	setUser(String user)	Empty implementation

Type	Interface	Compatible with Hadoop MapReduce?
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.

Type	Interface	Compatible with Hadoop MapReduce?
OutputCommitter	setupTask(TaskAttemptContext taskContext)	No. An exception is reported.
OutputCommitter	setupTask(TaskAttemptContext taskContext)	No. An exception is reported.
Counter	getDisplayName()	Yes
Counter	getName()	Yes
Counter	getValue()	Yes
Counter	increment(long incr)	Yes
Counter	setValue(long value)	Yes
Counter	setDisplayNames(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(URL uri, Configuration conf)	No. An exception is reported.
DistributedCache	addCacheFile(URL uri, Configuration conf)	No. An exception is reported.

Type	Interface	Compatible with Hadoop MapReduce?
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.
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.

Type	Interface	Compatible with Hadoop MapReduce?
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
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.
Multi{anchor:_GoBack}pleOutputs	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.

Type	Interface	Compatible with Hadoop MapReduce?
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.

8.4. Data types

MapReduce supports the following data types: bigint, double, string, datetime, boolean, decimal, tinyint, smallint, int, float, varchar, timestamp, binary, array, map, and struct. The following table lists the mappings between MaxCompute data types and Java types.

Data type mapping

MaxCompute type	Java type
Tinyint	java.lang.Byte
Smallint	java.lang.Short
Int	java.lang.Integer
Bigint	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Decimal	java.math.BigDecimal
Boolean	java.lang.Boolean
String	java.lang.String
Varchar	com.aliyun.odps.data.Varchar
Binary	com.aliyun.odps.data.Binary
Datetime	java.util.Date
Timestamp	java.sql.Timestamp
Array	java.util.List
Map	java.util.Map
Struct	com.aliyun.odps.data.Struct

8.5. Limits

The following limits apply to MapReduce:

- A Map or Reduce worker consumes 2,048 MB memory by default. The value range is 256 MB to 12 GB.
- Each task can reference up to 256 resources. Each partitioned table is considered as one unit.
- Each task can have up to 1,024 inputs and up to 256 outputs.
- Each task can have up to 64 custom counters.
- The number of Map instances in a job is calculated by the framework based on the split size. If no input table is specified, you can use `odps.stage.mapper.num` to set the number of Map instances. The number is in the range of 1 to 100,000.
- The default number of Reduce instances in a job is 1/4 of the number of Map instances. You can set this number in the range of 0 to 2,000. The following situation may occur: Reduce instances process much more data than Map instances, which results in a slow Reduce stage. Furthermore, a maximum of 2,000 Reduce instances can be created.
- Each Map or Reduce instance can retry three times after failure. Exceptions that do not allow retries can cause a job to fail.
- For local jobs, the number of Map or Reduce workers cannot exceed 100, while the number of downloads for an input is 100 by default.
- Each Map or Reduce worker can read a resource a maximum of 64 times.
- A task can reference a maximum of 2 GB resources.
- The framework determines the number of Map instances based on the split size.
- The length of string columns in MaxCompute tables cannot exceed 8 MB.
- If no data access operations or heartbeat packets are sent through `context.progress()`, the default timeout period of a Map or Reduce worker is 600s.

8.6. Sample programs

8.6.1. 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.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;
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();
    }
    @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);
    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);
    JobClient.runJob(job);
}
}

```

8.6.2. MapOnly example

For MapOnly jobs, Map directly sends < Key, Value > pairs to tables on MaxCompute. You only need to specify output tables. You do not need to specify the key-value metadata for map output.

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);
            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);
            if (is) {
                Record result = context.createOutputRecord();

```

```

    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);
    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);
    job.setNumReduceTasks(0);
    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]);
        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. Example: Input and output data to multiple objects

MaxCompute jobs can read data from multiple input tables, and write data to multiple output tables. All input tables for a job must have the same schema (number and type of columns). The output tables for a job can have different schemas (number and type of columns).

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.
 *
 *To run: jar -resources odps-mapred-example-0.12.0.jar com.aliyun.odps.mapred.open.example.
MultipleInOut
*mr_src,mr_src1,mr_srcpart|pt=1,mr_srcpart|pt=2/ds=2
*mr_multiinout_out1,mr_multiinout_out2|a=1/b=1|out1,mr_multiinout_out2|a=2/b=2|out2;
 *
 **/
public class MultipleInOut {
    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 });
        }
        @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;
            private Record result1;
            private Record result2;
            @Override
            public void setup(TaskContext context) throws IOException {
                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);
    // No label is specified. Default output is adopted.
    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, "out1");
    result2.set(1, 1L); context.write(result2, "out2");
}

public static LinkedHashMap<String, String> convertPartSpecToMap( String partSpec) {
    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(TokenizeMapper.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).label(ss[2]).build(), job);
} } else {
System.err.println("Style of output: " + out + " is not right");
System.exit(1);
}
}
JobClient.runJob(job);
}
}

```

8.6.4. Multi-task 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.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;
/**
 * MultiJobs
 *
 * Running multiple job
 *
 * To run: jar -resources > multijobs_res_table,odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.MultiJobs mr_multijobs_out;
 */
public class MultiJobs {
    public static class InitMapper extends MapperBase {
        @Override
        public void setup(TaskContext context) throws IOException { Record record = context.createOutputRecord();
        long v = context.getJobConf().getLong("multijobs.value", 2);
        record.set(0, v);
        context.write(record);
        }
    }
    public static class DecreaseMapper extends MapperBase {
        @Override
        public void cleanup(TaskContext context) throws IOException {
            // Obtain the variable values defined by the main function from JobConf.
            long expect = context.getJobConf().getLong("multijobs.expect.value", -1);
            long v = -1;
            int count = 0;
            Iterator<Record> iter = context.readResourceTable("multijobs_res_table");
            while (iter.hasNext()) {
                Record r = iter.next();
                v = (Long) r.get(0);
                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);
            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"));
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);
    decJob.setNumReduceTasks(0);
    RunningJob rJob = JobClient.runJob(decJob); iterCount--;
    if (rJob.getCounters().findCounter("multijobs", "value").getValue() == 0) { break;
    }
    }
    if (iterCount != 0) {
        throw new IOException("Job failed.");
    }
    }
}
```

8.6.5. Secondary sorting 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 by the first number.
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.SecondarySort mr_sort_in > mr_secondarysort_out;
```



```

*
**/
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: secondarysort <in> <out>");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(MapClass.class);
job.setReducerClass(ReduceClass.class);

```

```

job.setReducerClass(ReducerClass.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

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
 *
 * To run: jar -resources > odps-mapred-example-0.12.0.jar,mr_join_src1.txt
 * com.aliyun.odps.mapred.open.example.Upload mr_join_src1.txt > mr_join_src1;
 *
 */
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);
    job.set("import.filename", args[0]);
    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 the following methods to configure JobConf:

- Use the JobConf API in the SDK. This method has the highest priority.
- Use the -conf parameter in a jar command to specify a new JobConf file. This method has the lowest priority. For how to use -Conf, refer to the relevant command.

8.6.7. Example for using counters

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 and operate custom counters.

Example:

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException; import java.util.Iterator;

```

```

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
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.UserDefinedCounters mr_src > mr_testcounters_out;
 *
 */
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 });
        }
        @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);
    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

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.
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.Grep mr_src mr_grep_tmp > mr_grep_out val;
 */
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 = 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 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);
    }
    }
    /**
     * 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 IOException {
            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.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);
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);
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" });
// sort by
// decreasing
// freq
@SuppressWarnings("unused")
RunningJob rjSort = JobClient.runJob(sortJob);
}
}

```

8.6.9. JOIN example

The MaxCompute MapReduce framework does not support the JOIN logic. However, you can use custom Map and Reduce functions to complete JOIN operations. This requires extra work.

Table `mr_join_src1`(key bigint, value string) must be joined with `mr_join_src2`(key bigint, value string). The output table is `mr_join_out`(key bigint, value1 string, value2 string), where value1 indicates the value of `mr_join_src1` and value2 indicates the value of `mr_join_src2`.

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 com.aliyun.odps.counter.Counter;
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;

```



```

import com.aliyun.odps.mapred.examples.schemautil;
/**
 * Join
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.Join mr_join_src11 > mr_join_src22 mr_join_out;
 *
 */
public class Join {
    public static class JoinMapper extends MapperBase {
        private Record mapkey;
        private Record mapvalue;
        @Override
        public void setup(TaskContext context) throws IOException {
            mapkey = context.createMapOutputKeyRecord();
            mapvalue = context.createMapOutputValueRecord();
        }
        @Override
        public void map(long key, Record record, TaskContext context) throws IOException {
            /* Determine the source table of the record based on the value field. This is the user code
            logic. If the source table of the record cannot be determined based on the value field, you
            can add a field in the input table. The tag generated based on the value field is used in c
            onnection operations in the Reduce stage. */
            long tag = 1;
            String val = record.get(1).toString();
            if (val.startsWith("valb_")) {
                tag = 2;
            }
            mapkey.set(0, Long.parseLong(record.get(0).toString()));
            mapkey.set(1, tag);
            mapvalue.set(0, tag);
            for (int i = 1; i < record.getColumnCount(); i++) {
                mapvalue.set(i, 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();
        }
        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOExcep
            tion {
            long k = (Long) key.get(0);
            List<Object[]> list1 = new ArrayList<Object[]>();
            Counter cnt = context.getCounter("MyCounters", "reduce_outputs");
            cnt.increment(1);
            while (values.hasNext()) {
                Record value = values.next();
                long tag = (Long) value.get(0);
                if (tag == 1) {
                    //If the data comes from the first table, the data is cached in the list.

```

```

//Data is sorted by key and tag, so the value with tag==1 is sorted on the top.
//We recommended that you exercise caution when using this method in practice. If a key has
many values,
//the memory usage of Reduce stage increases. When the memory usage exceeds the value set b
y JobConf::setMemoryForReduceTask,
//the Reduce stage may be terminated by the system due to memory overflow.
list1.add(value.toArray().clone());
} else {
//If the data comes from the second table, the data is sorted by key and tag.
//All values in the first table have been saved in list1.
//For the values in the first table
for (Object r1: list1) { int index = 0;
//Set the key first.
result.set(index++, k);
Object[] s_arr = (Object[])r1;
result.set(index++, s_arr[1].toString());
result.set(index++, value.get(1).toString());
context.write(result);
}
}
}
}
}

public static void main(String[] args) throws Exception {
if (args.length != 3) {
System.err.println("Usage: Join <input table1> <input table2> <out>");
System.exit(2);
}
JobConf job = new JobConf();
job.setMapperClass(JoinMapper.class);
job.setReducerClass(JoinReducer.class);
job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,tag:bigint"));
job.setMapOutputValueSchema(SchemaUtils.fromString("tagx:bigint,value:string"));
job.setPartitionColumns(new String[] { "key" });
//Sort data by key and tag. The JOIN operation can be performed in the Reduce stage only af
ter data sorting by key.
//The tag indicates the source table of the current record.
job.setOutputKeySortColumns(new String[] { "key", "tag" });
job.setOutputGroupingColumns(new String[] { "key" });
//The Reduce stage uses lists to cache data. Therefore, we recommend that you increase the
memory size for Reduce workers.
job.setMemoryForReduceTask(4096);
job.setInt("table.counter", 0);
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
InputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), job);
JobClient.runJob(job);
}
}

```

8.6.10. Sleep example

Example:

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.OdpsException;
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;
/**
 * Dummy class for testing MR framefork. Sleeps for a defined period of > time in
 * mapper and reducer. Generates fake input for map / reduce jobs. Note > that
 * generated number of input pairs is in the order of
 * <code>numMappers
 * mapSleepTime / 100</code>, so the > job uses some disk
 * space.
 * To run: jar -resources odps-mapred-example-0.12.0.jar > com.aliyun.odps.mapred.open.examp
 * le.SleepJob
 * -m 1 -r 1 -mt 1 -rt 1;
 *
 */
public class SleepJob {
    private static Log LOG = LogFactory.getLog(SleepJob.class);
    public static class SleepMapper extends MapperBase {
        private LinkedHashMap<Integer, Integer> inputs = new LinkedHashMap<Integer, Integer>();
        private long mapSleepDuration = 100;
        private int mapSleepCount = 1;
        private int count = 0;
        private Record key;
        @Override
        public void setup(TaskContext context) throws IOException{
            LOG.info("map setup called");
            JobConf conf = (JobConf) context.getJobConf();
            mapSleepCount = conf.getInt("sleep.job.map.sleep.count", 1);
            if (mapSleepCount < 0)
                throw new IOException("Invalid map count: " + mapSleepCount);
            mapSleepDuration = conf.getLong("sleep.job.map.sleep.time", 100)
                / mapSleepCount;
            LOG.info("mapSleepCount = " + mapSleepCount + ", mapSleepDuration = " + mapSleepDuration);
            final int redcount = conf.getInt("sleep.job.reduce.sleep.count", 1);
            if (redcount < 0)
                throw new IOException("Invalid reduce count: " + redcount);
            final int emitPerMapTask = (redcount * conf.getNumReduceTasks());

```

```

int records = 0;
int emitCount = 0;
while (records++ < mapSleepCount) {
    int key = emitCount;
    int emit = emitPerMapTask / mapSleepCount;
    if ((emitPerMapTask) % mapSleepCount > records) {
        ++emit;
    }
    emitCount += emit;
    int value = emit;
    inputs.put(key, value);
}
key = context.createMapOutputKeyRecord();
}
@Override
public void cleanup(TaskContext context) throws IOException {
    // it is expected that every map processes mapSleepCount number of
    // records.
    LOG.info("map run called");
    for (Map.Entry<Integer, Integer> entry : inputs.entrySet()) {
        LOG.info("Sleeping... (" + mapSleepDuration * (mapSleepCount - count)) + ") ms left");
        try {
            Thread.sleep(mapSleepDuration);
        } catch (InterruptedException e) { throw new IOException(e); }
    }
    ++count;
    // output reduceSleepCount * numReduce number of random values, so that
    // each reducer will get reduceSleepCount number of keys.
    int k = entry.getKey();
    int v = entry.getValue();
    for (int i = 0; i < v; ++i) {
        key.set(new Object[] { (Long) ((long) (k + i)) });
        context.write(key, key);
    }
}
}

public static class SleepReducer extends ReducerBase {
    private long reduceSleepDuration = 100;
    private int reduceSleepCount = 1;
    private int count = 0;
    @Override
    public void setup(TaskContext context) throws IOException {
        LOG.info("reduce setup called");
        JobConf conf = (JobConf) context.getJobConf();
        reduceSleepCount = conf.getInt("sleep.job.reduce.sleep.count",
            reduceSleepCount);
        reduceSleepDuration = conf.getLong("sleep.job.reduce.sleep.time", 100)
            / reduceSleepCount;
        LOG.info("reduceSleepCount = " + reduceSleepCount
            + ", reduceSleepDuration = " + reduceSleepDuration);
    }
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOExcep
tion {

```

```

    }
    LOG.info("reduce called");
    LOG.info("Sleeping... ("
+ (reduceSleepDuration * (reduceSleepCount - count)) + ") ms left"); try {
    Thread.sleep(reduceSleepDuration);
    } catch (InterruptedException e) {
    throw new IOException(e);
    }
    count++;
    }
    }

    public static int run(int numMapper, int numReducer, long mapSleepTime, int mapSleepCount,
    long reduceSleepTime, int reduceSleepCount)
    throws OdpsException {
    JobConf job = setupJobConf(numMapper, numReducer, mapSleepTime, mapSleepCount, reduceSleepT
ime, reduceSleepCount);
    JobClient.runJob(job);
    return 0;
    }

    public static JobConf setupJobConf(int numMapper, int numReducer, long mapSleepTime, int ma
pSleepCount, long reduceSleepTime, int reduceSleepCount) {
    JobConf job = new JobConf();
    InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName("mr_sleep_out").build(), job);
    job.setNumReduceTasks(numReducer);
    job.setMapperClass(SleepMapper.class);
    job.setReducerClass(SleepReducer.class);
    job.setMapOutputKeySchema (SchemaUtils.fromString("int1:bigint"));
    job.setMapOutputValueSchema (SchemaUtils.fromString("int2:bigint"));
    job.setPartitionColumns(new String[] { "int1" });
    job.setLong("sleep.job.map.sleep.time", mapSleepTime);
    job.setLong("sleep.job.reduce.sleep.time", reduceSleepTime);
    job.setInt("sleep.job.map.sleep.count", mapSleepCount);
    job.setInt("sleep.job.reduce.sleep.count", reduceSleepCount);
    return job;
    }

    private static void printUsage() {
    System.err.println("SleepJob [-m numMapper] [-r numReducer]"
+ " [-mt mapSleepTime (msec)] [-rt reduceSleepTime (msec)]"
+ " [-recordt recordSleepTime (msec)]");
    }

    public static void main(String[] args) throws Exception {
    if (args.length < 1) {
    printUsage();
    return;
    }

    int numMapper = 1, numReducer = 1;
    long mapSleepTime = 100, reduceSleepTime = 100, recSleepTime = 100;
    int mapSleepCount = 1, reduceSleepCount = 1;
    for (int i = 0; i < args.length; i++) { if (args[i].equals("-m")) {
    numMapper = Integer.parseInt(args[++i]);
    } else if (args[i].equals("-r")) {
    numReducer = Integer.parseInt(args[++i]);
    } else if (args[i].equals("-mt")) {
    mapSleepTime = Long.parseLong(args[++i]);

```

```

    } else if (args[i].equals("-rt")) {
        reduceSleepTime = Long.parseLong(args[++i]);
    } else if (args[i].equals("-recordt")) { recSleepTime = Long.parseLong(args[++i]);
    }
}

// sleep for *SleepTime duration in Task by recSleepTime per record
mapSleepCount = (int) Math.ceil(mapSleepTime / ((double) recSleepTime));
reduceSleepCount = (int) Math.ceil(reduceSleepTime
/ ((double) recSleepTime));
run(numMapper, numReducer, mapSleepTime, mapSleepCount, reduceSleepTime, reduceSleepCount);
}
}

```

8.6.11. 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
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.open.mapred.example.Unique mr_sort_in > mr_unique_out
 * key|value|all;
 *
 */
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 IOExcep
tion {
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];
}
// 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

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.
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar > com.aliyun.odps.mapred.open.examp
le.Sort
 * mr_sort_in mr_sort_out;
 *
 */
public class Sort {

```



```

public class Sort {
    static int printUsage() {
        System.out.println("sort <input> <output>"); return -1;
    }
    /**
     * Implement 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);
        jobConf.setNumReduceTasks(1);
        Jobconf.setmapoutputkeyschema schemautils schemeiutils.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 end_time = new Date(); System.out.println("Job ended: " + end_time); System.out.printl
n("The job took " + (end_time.getTime() - startTime.getTime()) / 1000 + " seconds." );
    }
}

```

8.6.13. Example of using partitioned table as an input

The following examples use partitions as an 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 the preceding example, the MaxCompute SDK and MapReduce SDK are used together to allow MapReduce tasks to read data from partitions. The preceding code cannot be compiled for execution. It is just an example of the main function. In the preceding example, the applicable function is the user logic that determines whether the partitions can be used as an input of the MapReduce job.

8.6.14. Pipeline example

Example:

```
package com.aliyun.odps.mapred.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 Histogram {
    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 num;
        private Record result;
        @Override
        public void setup(TaskContext context) throws IOException {
            num = context.createOutputKeyRecord();
            result = context.createOutputValueRecord();
        }
        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOExcep
```

```

    tion {
    long count = 0;
    while (values.hasNext()) {
    Record val = values.next();
    count += (Long) val.get(0);
    }
    result.set(0, key.get(0));
    num.set(0, count);
    context.write(num, result);
    }
    }
    public static class IdentityReducer extends ReducerBase {
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOExcep
    tion {
    while (values.hasNext()) {
    context.write(values.next());
    }
    }
    }
    public static void main(String[] args) throws OdpsException {
    if (args.length != 2) {
    System.err.println("Usage: orderedwordcount <in_table> <out_table>");
    System.exit(2);
    }
    Job job = new Job();
    /**
    * In the process of constructing pipeline, if you do not specify mapper's OutputKeySortColu
    mns, PartitionColumns, OutputGroupingColumns,
    * the framework defaults to its OutputKey as the default configuration for the three
    ***/
    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("count", OdpsType.BIGINT) })
    .setOutputValueSchema(
    new Column[] { new Column("word", OdpsType.STRING)})
    .addReducer(IdentityReducer.class).createPipeline();
    job.setpipeline(pipeline);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    job.submit(); job.waitForCompletion();
    System.exit(job.isSuccessful() == true ? 0 : 1);
    }
    }

```

9. MaxCompute Graph

9.1. Graph overview

9.1.1. Graph overview

MaxCompute Graph is a processing framework designed for iterative graph computing. Graph computing jobs use graphs to build models. Graphs are composed of vertices and edges with values. MaxCompute Graph supports the following operations to edit a graph:

- Editing the value of vertex or edge.
- Adding/deleting vertex.
- Adding/deleting edge.

 **Note** When editing the vertex or edge, you must maintain the relationship between the two items.

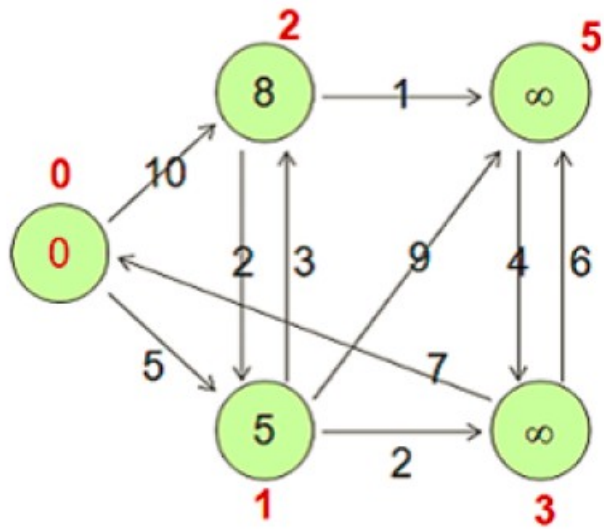
After performing iterative graph editing and evolution, you can get the final result. Typical applications include [PageRank](#), [SSSP algorithm](#), and [K-Means algorithm](#). Furthermore, you can use the Java SDK provided by MaxCompute Graph to compile computing programs.

9.1.2. Graph data structure

MaxCompute Graph processes directed graphs, or digraphs, that consist of a vertex and an edge. MaxCompute stores data in two-dimensional tables, so you must convert graph data into two-dimensional tables and store them in MaxCompute. To perform graph analysis, you must use a custom GraphLoader to convert two-dimensional table data to vertexes and edges in the MaxCompute Graph engine. You can then determine how to break down and analyze your graph data based on your business requirements. In the following chapter, the examples provided use different tabular expressions to represent the data structure of a graph.

The vertex structure can be expressed as < ID, Value, Halted, Edges >, indicating respectively the vertex identifier, the value, the state (Halted, meaning whether to stop iteration), and the edge set (Edges, indicating lists of all edges starting from the vertex). The edge structure can be described as < DestVertexID, Value >, indicating respectively the destination vertex (DestVertexID) and value (Value). The following figure shows Graph data structure.

Graph data structure



The preceding figure involves the following vertexes.

Graph data structure

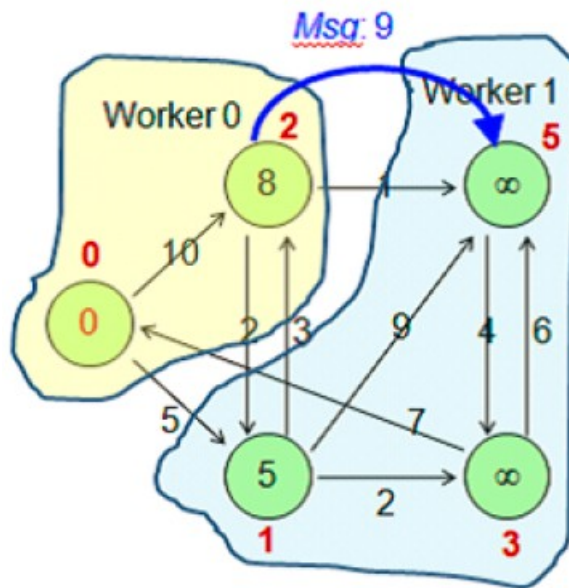
Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

9.1.3. Graph logic

9.1.3.1. Load graph

- Graph load: The framework calls the custom GraphLoader to parse the records in the input table into vertices or edges.
- Partitioning: The framework calls the custom Partitioner to partition vertices (default partition logic: the hash value of the vertex ID modulo the number of workers). Then, the framework distributes the partitions to the corresponding workers.

Load graph



Based on the preceding figure, if there are two workers, v0 and v2 are distributed to Worker0, because the result of the ID modulo 2 (total number of workers) is 0. v1, v3, and v5 are distributed to Worker1, because the result of the ID modulo 2 is 1.

9.1.3.2. Iterative computation

An iteration is a superstep. During a superstep, all vertices not in the halted state (Halted value is false) or vertices that receive messages (vertices in the halted state automatically wake up when receiving a message) are traversed. The compute method (ComputeContext context, Iterable messages) of these vertices is called.

In a custom compute method (ComputeContext context, Iterable messages):

- The messages sent by the previous superstep to the current vertex are processed.
- The graph is edited as required:
 - The values of vertices or edges are modified.
 - Messages are sent to some vertices.
 - Vertices or edges are added or deleted.
- The aggregator aggregates information to obtain global information.
- The current vertex is set to the halted or non-halted state.
- During each iteration, the framework automatically sends messages to the corresponding workers asynchronously. The messages are processed in the next superstep.

9.1.3.3. End of iteration

An iteration ends if any of the following conditions is satisfied.

- All vertices are in the halted state (Halted value 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 overview

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 summarize and process global information.

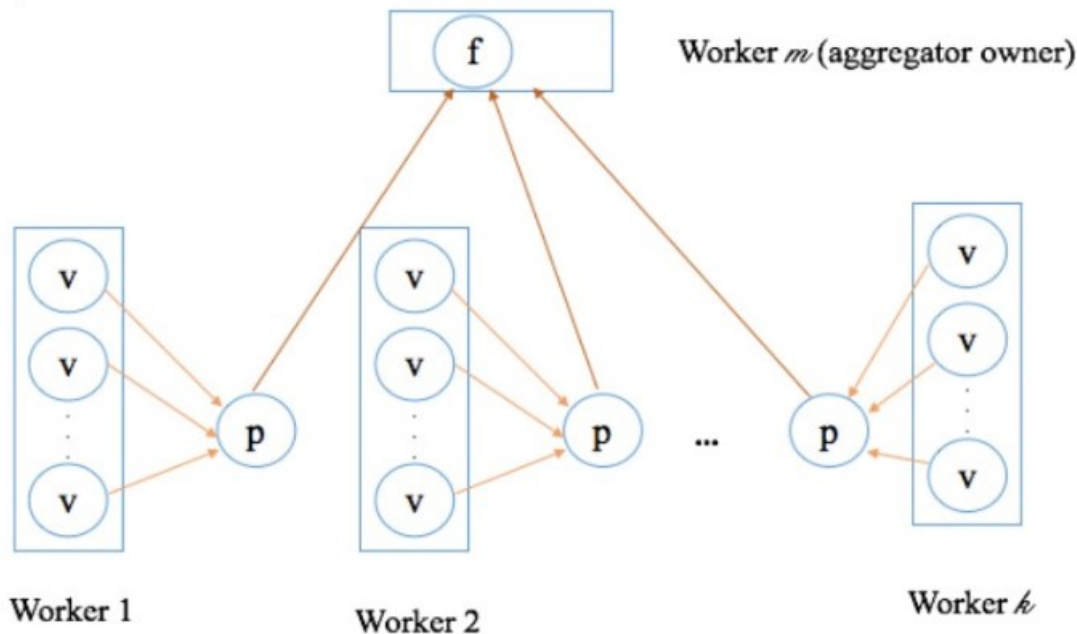
This topic describes the implementation mechanism and related API operations of Aggregator and uses Kmeans Clustering as an example to demonstrate how to use Aggregator.

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 all supersteps start. It is used to initialize AggregatorValue. In the first superstep iteration (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 or Owner is located.

- **merge(value, partial)**

This API operation is performed on the Worker where Aggregator or Owner is located. 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, while `partial` indicates objects to be aggregated. `partial` is used to update value.

Assume that there are three Workers `w0`, `w1`, and `w2`, and the corresponding partial aggregation results are `p0`, `p1`, and `p2`. If `p1`, `p0`, and `p2` are sent to the Worker where Aggregator or Owner is located in sequence, the following merge operations are performed:

- i. `merge(p1, p0)` is executed first to aggregate `p1` and `p0` as `p1`.
- ii. Then, `merge(p1, p2)` is executed to aggregate `p1` and `p2` as `p1`. `p1` is the global aggregation result in this superstep.

Therefore, when only one Worker exists, the merge method is not required. In this case, `merge()` is not called.

- **terminate(context, value)**

After the Worker where Aggregator or Owner is located executes `merge()`, the framework calls `terminate(context, value)` to perform the final processing. The second parameter value indicates the global aggregation result obtained by calling `merge()`. The global aggregation result can be further modified in this API operation. After `terminate()` is executed, the framework distributes the global aggregation object to all Workers for the next superstep.

If `true` is returned for `terminate()`, iteration is ended for the entire job. Otherwise, the iteration continues. In machine learning scenarios, jobs end generally when `true` is returned after convergence.

Kmeans Clustering example

This section uses typical Kmeans Clustering as an example to demonstrate how to use Aggregator.

 **Note** If you need 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 Kmeans 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 closest to a specific center. For example, `sums(i,j)` indicates the sum of dimension j of the sample 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 resource file `centers`, 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 then 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 global aggregation operations performed on the Worker where Aggregator Owner is located are described as follows:

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 sum up 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" + n
ewCentroids);
    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. `isConverged()` is then called to determine 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(false)` is set to `false`, data loaded by each Worker is not partitioned based on `Partitioner`. Data is maintained by the Worker that loads it.

Summary

The basic steps of Aggregator are as follows:

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 Aggregator Owner is located.
5. The Worker where Aggregator Owner is located executes merge multiple times to implement global aggregation.
6. The Worker where Aggregator Owner is located 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

The MaxCompute client provides a jar command for running MaxCompute Graph jobs. This command is used in the same way as the jar command in MapReduce.

Command 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, separate by comma
```

The following table describes the parameters.

Parameters

Parameter	Description
-conf <configuration file>	Indicates a JobConf file.
-classpath <local_file_list>	Indicates the classpath for local execution. It specifies the local paths (including relative path and absolute path) of the JAR package where the main function is located.
-D <prop_name>=<prop_value>	Indicates the Java attribute of <mainClass> in local execution. You can define multiple attributes.
-local	Runs the MapReduce job locally, mainly for program debugging.
-resources <resource_name_list>	Declares the resources used for running the Graph job. You typically need to specify the name of the resource where the Graph job is located in resource_name_list.  Notice If you read other MaxCompute resources while running the Graph job, you also need to add those resource names to resource_name_list. Multiple resources must be separated by commas (.). If you need to use resources of another project, add the prefix PROJECT_NAME/resources/, for example, -resources otherproject/resources/resfile.

 **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 through the MaxCompute client. Take the PageRank algorithm as an example.

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 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 resources corre
    // spond to those specified by -libjars in the jar command.
    job.addCacheResourceToClassPath("mapreduce-examples.jar");
    // Set the configuration item corresponding to odps_config.ini in the client. Replace it wi
    // th 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

The input and output of MaxCompute Graph jobs must be tables. You cannot customize the input or output format.

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 col2 and col0 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

- Multiple inputs are supported.
- Partition filtering is not supported. For more application limits, see [Application limits](#).
- For more information about job input definition, see the `addInput` method description in `GraphJob`.
- The framework reads records from the input table and transfers the records to the user-defined `GraphLoader` to load graph data.

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 will overwrite partitions specified in tableinfo, which is similar to the INSERT OVERWRITE operation. False is similar to the INSERT INTO operation.
```

 Note

- Multiple outputs are supported, and each output is identified by a label.
- A Graph job can use the `Write` method of `WorkContext` to write data to an output table during runtime. Multiple outputs must be labeled.
- For more information about job output definition, see the `addOutput` method description in `GraphJob`.

9.2.3. Read data from resources

9.2.3.1. Add resource in Graph program

In addition to the `jar` command, the following two methods of `GraphJob` can be used to specify the resources read by Graph:

```
void addCacheResources(String resourceNames)
void addCacheResourcesToClassPath(String resourceNames)
```

9.2.3.2. Use resources in Graph

In Graph, you can use the following methods of the corresponding context object `WorkerContext` to read resources:

```

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;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName, String relativePath) throws IOException;

```

Note

- Normally, resources are read in the setup of WorkerComputer, saved in WorkerValue, and obtained through getWorkerValue.
- The preceding stream API is recommended while reading and processing to reduce memory consumption.
- For more information about limits, see [Application limits](#).

9.3. Graph SDK introduction

Major APIs

API	Description
GraphJob	GraphJob is inherited from JobConf to define, submit, and manage a MaxCompute Graph job.
Vertex	Vertex is an abstract of a graph and has the following attributes: id, value, halted, and edges. It is implemented through the setVertexClass API in GraphJob.
Edge	Edge is an abstract of a graph and has the following attributes: destVertexId and value. The graph data structure is maintained by an adjacency list. The outgoing edges of a vertex are stored in its edges attribute.
GraphLoader	GraphLoader is used to load graphs. It is set through the setGraphLoaderClass API in GraphJob.
VertexResolver	VertexResolver is used to customize the collision processing logic of the revising graph topology. It provides this logic through the setLoadingVertexResolverClass and setComputingVertexResolverClass APIs in GraphJob for graph loading and iteration computing.
Partitioner	Partitioner is used to partition graphs for partition computing. It is set through the setPartitionerClass API in GraphJob. By default, the HashPartitioner is used to first obtain the Vertex ID Hash value, and then to model the number of Workers.
WorkerComputer	WorkerComputer allows customized logic to be executed while Worker starts and exits. WorkerComputer is set through the setWorkerComputerClass API in GraphJob.

API	Description
Aggregator	Allows you to define one or multiple Aggregators through the <code>setAggregatorClass(Class ...)</code> API in Aggregator.
Combiner	You can set Combiner through the <code>setCombinerClass</code> API in Combiner.
Counters	In the job operating logic, counters can be taken and counted through the <code>WorkerContext</code> API, and the framework will automatically summarize them.
WorkerContext	Context objects encapsulate the functions provided by the framework, such as revising the topology of graphs, sending messages, writing results, reading resources, and so on.

9.4. Development and debugging

9.4.1. Development procedure

MaxCompute does not provide plug-ins for Graph development. Instead, you can develop MaxCompute Graph programs in Eclipse. The recommended development procedure is as follows:

1. Write Graph code and perform local debugging to test basic functions.
2. Perform cluster debugging to verify results.

9.4.2. Development example

Context

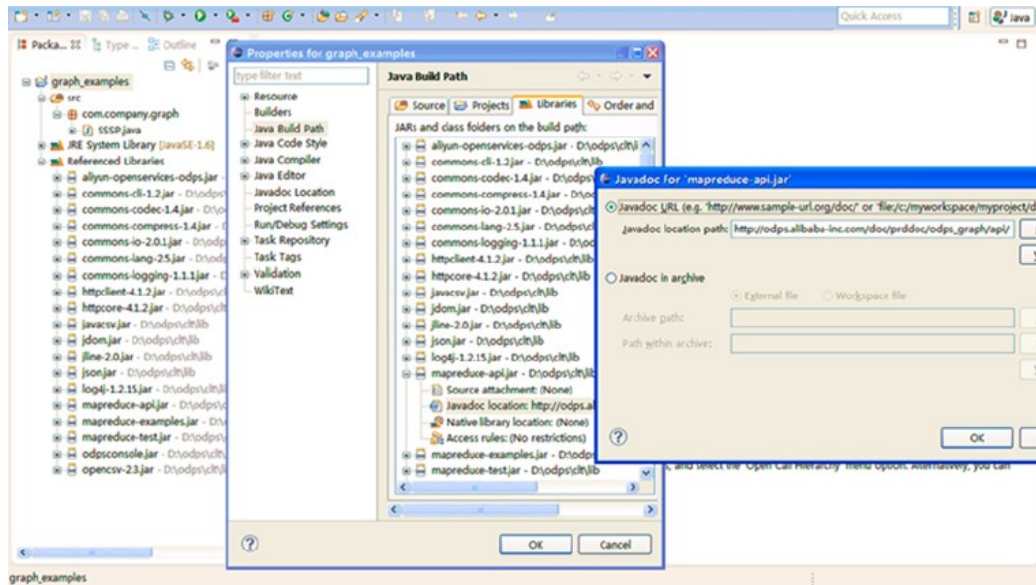
This topic uses the SSSP algorithm as an example to describe how to use Eclipse to develop and debug a Graph program. The development procedure is as follows:

Procedure

1. Create a Java project.

In this example, the project is `graph_examples`. Add the JAR package in the `lib` directory of the MaxCompute client to Build Path of the Eclipse project. The following figure shows a configured Eclipse project.

Create a Java project



2. Develop a MaxCompute Graph program.

In the actual development process, you can copy a sample program (such as SSSP) and then modify it as required. In this example, only the package path is changed to *package com.aliyun.odps.graph.example*.

3. Compile and build the package. In an Eclipse environment, right-click the source code directory (the `src` directory in the figure) and choose **Export** > **Java** > **JAR file** to generate a JAR package. Select the path for storing the target JAR package, such as *D:\odps\clt\odps-graph-example-sssp.jar*.
4. Use the MaxCompute client to run SSSP. For more information, see [Compile and run a Graph job](#).

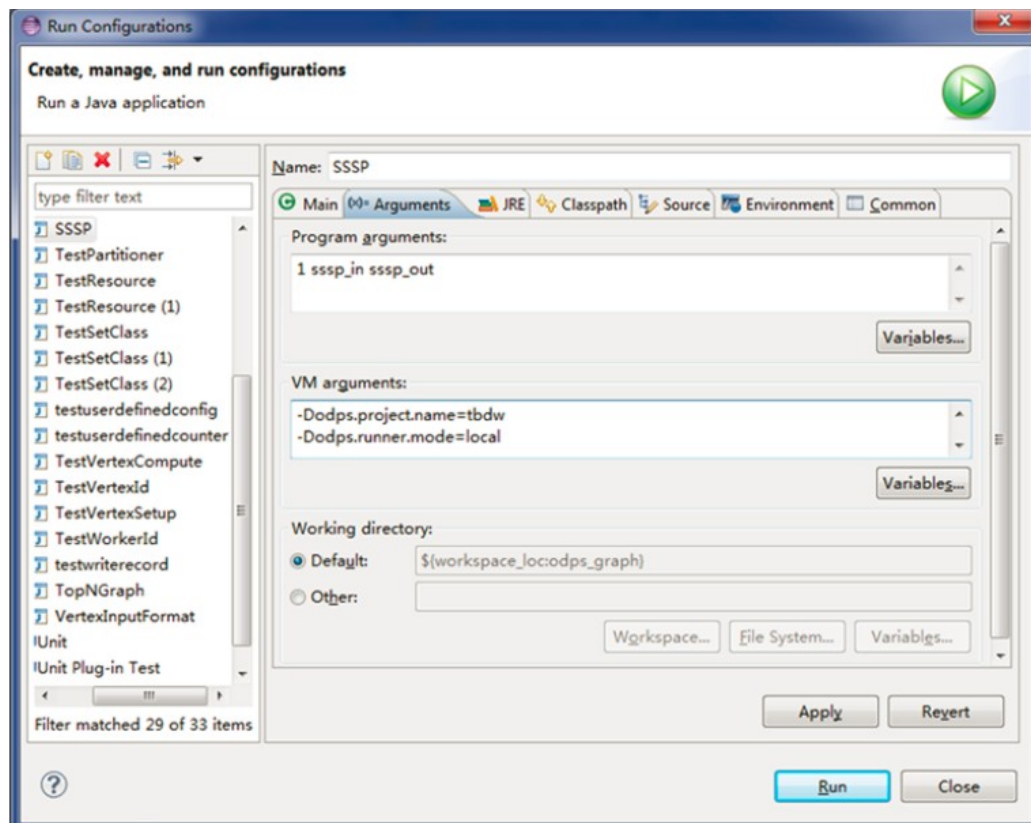
9.4.3. Local debugging

MaxCompute Graph supports the local debugging mode. You can use Eclipse for breakpoint debugging. The breakpoint debugging procedure is as follows:

Procedure

1. Select an Eclipse project. Right-click the Graph job main program file (the file that contains the main function), and configure its execution arguments, as shown in the following figure.

Local debugging



2. On the **Arguments** tab page, set the following arguments as the input parameters of the main program:
 - Program arguments: 1 sssp_in sssp_out.
 - VM arguments: Dodps.runner.mode=local, Dodps.project.name=<project.name>, Dodps.end.point=<end.point>, Dodps.access.id=<access.id>, and Dodps.access.key=<access.key>.
 - For the local mode (odps.end.point not specified), you need to create the sssp_in and sssp_out tables in the warehouse, and add the following data to sssp_in:

```
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, see [Run MapReduce tasks locally](#).

3. Click **Run** to run SSSP on the local machine.

Refer to conf/odps_config.ini in the MaxCompute client for the settings of common parameters. The other parameters are described as follows:

- odps.runner.mode: The value is local. It is required for the local debugging feature.
- odps.project.name: specifies the current project, which is required.
- odps.end.point: specifies the current MaxCompute service address, which is optional. If this parameter is not specified, SSSP only reads metadata and data from the tables or resources in

the warehouse. If the data does not exist, an error is returned. If this parameter is specified, SSSP first reads data from the warehouse. If the data does not exist, it reads data from the remote MaxCompute service.

- `odps.access.id`: specifies the AccessKey ID for accessing the MaxCompute service. It is valid only if `odps.end.point` is specified.
- `odps.access.key`: specifies the AccessKey secret for accessing the MaxCompute service. It is effective only if `odps.end.point` is specified.
- `odps.cache.resources`: specifies the resource list to be used. This parameter is the same as `-resources` of the `jar` command.
- `odps.local.warehouse`: specifies the local warehouse path. It is `./warehouse` by default.

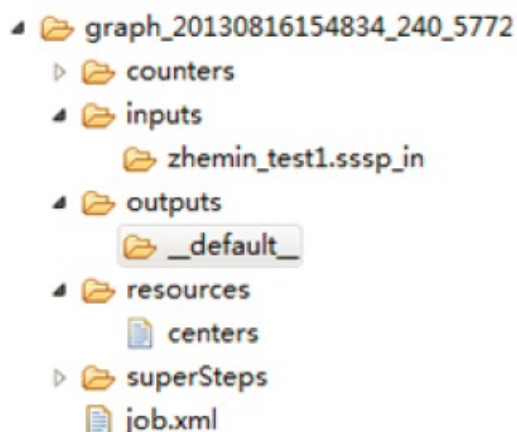
The output is as follows:

```
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 the preceding examples, the local warehouse must contain the `sssp_in` and `sssp_out` tables. For more information about `sssp_in` and `sssp_out`, see [Compile and run Graph](#).

9.4.4. Temporary directory for local jobs

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 for a local Graph job contains the following directories and files:

- `counters`: stores the counter information that is generated during the running of the job.
- `inputs`: stores the input data of the job. Graph first reads data from the local warehouse. If no data is found, Graph uses the MaxCompute SDK to read data from the server (if `odps.end.point` is configured). An input operation reads 10 records by default. You can use the `Dodps.mapred.local.record.limit` parameter to modify the number of records read during each input operation. Up to 10,000 records can be read each time.

- **outputs**: stores the output data of the job. If there is an output table in the local warehouse, the results in outputs will overwrite data in that table after the job is executed.
- **resources**: stores the resources used by the job. Similarly, Graph first reads data from the local warehouse. If no data is found, Graph uses the MaxCompute SDK to read data from the server (if `odps.endpoint` is configured).
- **job.xml**: stores job configurations.
- **superstep**: stores persistent message information from each iteration.

 **Notice** If you need to output 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

After you perform local debugging, you can perform the following steps to submit a job for cluster 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 execute the job and check the operation log and command output.

 **Notice** For more information about running a Graph job in a cluster, see [Compile and run a Graph job](#).

9.4.6. Performance optimization

9.4.6.1. Configure job parameters

The following table lists the GraphJob configuration parameters that affect job performance.

GraphJob configuration parameters

Parameter	Description
<code>setSplitSize(long)</code>	Indicates the split size in MB. The value must be greater than 0. Default value: 64.
<code>setNumWorkers(int)</code>	Indicates the number of job workers. Value range: 1 to 1,000. Default value: 1. The number of workers is determined by the number of input bytes and split size.
<code>setWorkerCPU(int)</code>	Indicates the CPU resources for a map job. 100 resources are equivalent to one CPU core. Value range: 50 to 800. Default value: 200.
<code>setWorkerMemory(int)</code>	Indicates the memory resources in MB for a map job. Value range: 256M to 12G. Default value: 4,096.
<code>setMaxIteration(int)</code>	Indicates the maximum number of iterations. Default value: -1. If the value is equal to or smaller than 0, the job is not terminated after the maximum number of iterations.

Parameter	Description
<code>setJobPriority(int)</code>	Indicates the job priority. Value range: 0 to 9. Default value: 9. A greater value indicates a lower priority.

Recommendations:

1. Use `setNumWorkers` to increase the number of workers.
2. Use `setSplitSize` to reduce the split size and increase the data loading speed.
3. Use `setWorkerCPU` or `setWorkerMemory` to increase the CPU or memory resources for workers.
4. Use `setMaxIteration` to set the maximum number of iterations. For applications that do not require precise results, you can reduce the number of iterations to accelerate the iterating process.

Use `setNumWorkers` and `setSplitSize` together to accelerate data loading. If `setNumWorkers` is `workerNum`, `setSplitSize` is `splitSize`, and the total number of input bytes is `inputSize`, `splitNum` equals `inputSize` divided by `splitSize`. The relationship between `workerNum` and `splitNum` is as follows:

1. If `splitNum` is equal to `workerNum`, each worker loads one split.
2. If `splitNum` is greater than `workerNum`, each worker loads one or more splits.
3. If `splitNum` is smaller than `workerNum`, each worker loads zero or one split.

Therefore, you can adjust `workerNum` and `splitSize` to obtain a suitable loading speed. In the first two cases, data loading is faster. In the iteration phase, you only need to adjust `workerNum`. If you set runtime partitioning to false, we recommend that you either use `setSplitSize` to adjust the number of workers, or ensure the conditions in the first two cases are met. In the third case, the number of vertices in some of the workers is 0. In this case, you can use `odps.graph.split.size=<m>; set odps.graph.worker.num=<n>;` before the JAR command, which achieves the same effect as `setNumWorkers` and `setSplitSize`.

Another common performance problem is data skew. As indicated by the counters, some workers process much more vertices or edges than others.

Data skew usually occurs when the number of vertices, edges, or messages corresponding to certain keys is far greater than that corresponding to other keys. These keys are distributed for processing by a small number of workers, resulting in long execution time of these workers. You can use the following methods to resolve this issue:

- Use Combiner to aggregate the messages of vertices corresponding to the keys, to reduce the number of generated messages.
- Improve the business logic.

9.4.6.2. Use Combiner

Developers can set Combiner to reduce the memory and network traffic consumed by message storage, and reduce job execution time. For more information, see the Combiner description in [Graph SDK overview](#).

9.4.6.3. Reduce data input

If a disk stores large volumes of data, reading data from the disk may prolong the processing time. You can reduce the number of bytes to be read to increase the overall throughput and improve job performance. You can use either of the following methods:

- **Reduce data input:** For some decision-making applications, processing sampled data only affects the precision of the results, not the overall accuracy. In this case, you can sample specific data to the input table for further processing.
- **Avoid reading unnecessary fields:** The `TableInfo` class provided in the MaxCompute Graph framework can read specific columns (sent through a column name array), instead of the entire table or partition. This effectively reduces the amount of input data and improves job performance.

9.4.6.4. JAR packages

The following JAR packages are loaded on the JVM that runs Graph programs by default. 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



Notice In CLASSPATH of the running JVM, the preceding JAR packages are loaded before your JAR package. This may cause version conflicts. For example, your program calls a certain class function of `commons-codec-1.5.jar`, but the function is not included in the current MaxCompute packages. In this case, you can choose to call a similar function in version 1.3 or wait until MaxCompute is upgraded to the required version.

9.5. Application limits

- Each job can reference up to 256 resources. Each table or archive is considered as one unit.
- The total resource size referenced by a job cannot exceed 512 MB.
- Each job can have up to 1,024 inputs (the number of input tables cannot exceed 64). Each job can have up to 256 outputs.
- Labels specified for multiple outputs cannot be null or empty strings. The label length cannot exceed 256, and the label can contain only upper-case letters (A to Z), lower-case letters (a to z), digits (0 to 9), underlines (`_`), pound signs (`#`), periods (`.`), and hyphens (`-`).
- The number of custom counters in a job cannot exceed 64. The counter group name and counter name cannot contain pound signs (`#`), and the total length of both names cannot exceed 100.
- The number of workers for each job is calculated by the framework. The maximum number of workers is 1,000. An error is returned when the number of workers exceeds this value.
- Each worker consumes 200 units of CPU resources by default. The range of resources consumed is 50 to 800.
- Each worker consumes 4,096 MB memory by default. The range of memory consumed is 256 MB to 12

GB.

- Each worker can read from a single resource up to 64 times.
- The split size is 64 MB by default, but can be defined by the user. The split size must be greater than 0, and the maximum value is in the range of 20 to 9223372036854775807.
- GraphLoader/Vertex/Aggregator in MaxCompute Graph are restricted by Java Sandbox (however, the main program of a Graph job is not subject to this restriction) while they run in a cluster. For more information, see [Java sandbox restrictions](#).

9.6. Sample programs

9.6.1. SSSP

Dijkstra's algorithm is a typical algorithm for calculating the Single Source Shortest Path (SSSP) in a directed graph.

Shortest path: For a weighted directed graph $G = (V, E)$, many 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 algorithm is implemented as follows:

- Initialization: The distance from s to s is 0 ($d[s] = 0$), and the distance from u to s is infinite ($d[u] = \infty$).
- Iteration: If an edge from u to v exists, the shortest distance from s to v is updated to $d[v] = \min(d[v], d[u] + \text{weight}(u, v))$. The iteration ends until the distance from all vertices to s does not change.

 **Note** The implementation process determines that the algorithm is applicable to the MaxCompute Graph program. Each vertex maintains the current shortest distance to the source vertex. If the value changes, a message containing the new value and the edge weight is sent to the adjacent vertices. In the next iteration, the adjacent vertices update the current shortest distance based on the received message. The iteration ends when the current shortest distance values of all vertices do not change.

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 distance from this vertex to 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 distance to the source vertex to the result table.
```

```

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<Lo
    ngWritable> 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 vertex enters the halted state. The calculation ends when all vertices enter the halte
    d state.
    }
    }
    @Override
    public void cleanup(
    WorkerContext<LongWritable, LongWritable, LongWritable, LongWritable> context) throws IOExc
    eption {
    context.write(getId(), getValue());
    }
    }
    /** Define MinLongCombiner and combine messages sent to the same vertex to optimize perform
    ance and reduce memory usage.**/
    public static class MinLongCombiner extends
    Combiner<LongWritable, LongWritable> {
    @Override
    public void combine(LongWritable vertexId, LongWritable combinedMessage, LongWritable messa
    geToCombine) 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 vertex. The first column of the record is the vertex ID, and the second column stores all edge sets starting 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/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

PageRank is an algorithm for Web page ranking. For more information, see [PageRank](#). The input of the algorithm is a digraph G , where Vertex represents pages. If there is a link between page A to page B, there is an Edge linking A and B. Basic principles of the algorithm are as follows:

- Initialization: Vertex value means rank value of PageRank (double type). Initially, the value of all Vertices is $1/\text{TotalNumVertices}$.
- Iteration formula: $\text{PageRank}(i) = 0.15/\text{TotalNumVertices} + 0.85 * \text{sum}$. Sum indicates the sum of $\text{PageRank}(j)/\text{out\_degree}(j)$. (j indicates all vertices pointing to vertex i .)

 **Note** The PageRank algorithm is best suited to be run on MaxCompute Graph as each vertex maintains its PageRank value and sends PageRank(j)/out_degree(j) to its adjacent vertex (to vote it) per iteration. Upon the next iteration, each vertex re-calculates the PageRank value 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);
    /**
     * Defines PageRankVertex, where:
     * The vertex value indicates the current PageRank value of the vertex (web page).
     * The compute() method uses the iteration formula PageRank(i)=0.15/TotalNumVertices+0.85*sum
     * to update the vertex value.
     * The cleanup() method writes the vertex 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<Double
            Writable> 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 * sum);
                setValue(vertexValue);
            }
            if (hasEdges()) {
                context.sendMessageToNeighbors(this, new DoubleWritable(getValue().
                get() / getEdges().size()));
            }
        }
        @Override
        public void cleanup(
```

```

WorkerContext<Text, DoubleWritable, NullWritable, DoubleWritable> context) throws IOExcepti
on {
    context.write(getId(), getValue());
}
}

/** Define the PageRankVertexReader class, load a graph, and resolve each record in the tab
le into a vertex. The first column of the record is the start vertex and other columns are
the destination vertices.**/
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 IOExcep
tion {
        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 edgs 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 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

K-means clustering is a basic macro-clustering algorithm. Basic principles of the K-means clustering algorithm are as follows: Clustering is performed around k points in space, and the closest vertices are classified. The values of the clustering centers are successively updated through iterations until the optimal clustering result is obtained.

Assuming the sample set is divided into k sets or categories, the steps in the algorithm are as follows:

- Select initial center of k classes.
- In the i th iteration, select any sample, solve its path to k center, and then classify the sample into the class of shortest path to center.
- Use the mean method to update the center value of the class.
- For all k clustering centers, if the value remains unchanged or is less than a certain threshold after iteration of the first two steps, 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 KmeansVertex. The compute() method is simple. It calls the aggregate() method of
    the context object and transmits the value of the current vertex (in 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 vertex ID does not matter, and transmitted recordNum is used as the ID. The vertex value is the Tuple consisting of all columns of 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 {
```



```

, 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();
}
}
/**
 * Defines KmeansAggregator. This class encapsulates the main logic of the Kmeans algorithm,
 * where,
 * createInitialValue creates an initial value for each iteration (k-class center point). In
 * first iteration (superstep equals to 0), the value is the initial center point. Otherwise,
 * the value is the new center point when the last iteration ends.
 * The aggregate() method calculates the distance from each vertex to centers of different c
 * lasses, classifies the vertex as 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 the new central point based on the sum and count of eac
 * h class. If the distance between the new and old central points is less than a threshold va
 * lue or the number of iterations reaches the upper limit, the iteration ends (false is retur
 * ned). The final central point is written to the resulting 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 (glasvalue, 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, or Aggreg
ator, the maximum number of iterations (30 by default), and 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);
    // Specify job.setRuntimePartitioning(false). For the K-means algorithm, vertices do not ne
    ed to be distributed during graph loading. If RuntimePartitioning is set to false, the perf
    ormance for graph loading is improved.
    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 + " s
    econds");
}
}

```

9.6.4. BiPartiteMatching

In a bipartite graph, all vertices can be divided into two sets, to which the two vertices of each edge respectively belong. For a bipartite graph G , M is its subgraph. If any two edges of M 's edge set are not attached to the same vertex, then M is a match. The bipartite graph matching is usually used for information matching in scenarios with clear supply and demand relationships (such as online dating websites).

The procedure is as follows:

- Start from the first vertex on the left, select unmatched vertex to search for augmented path.
- If it goes through an unmatched vertex, the search is successful.
- Update path information, match number of Edge +1, and stop searching.
- If no augmented path is found, it does not search again from the specific vertex.

Example:

```

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

```

```

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, TextP
            air, NullWritable, 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  
        IOException {  
        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 component

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. This algorithm example is based on the parallel coloring algorithm.

Each vertex contains the following two parts:

- **colorID**: stores the color of the vertex (v) during forward traversal. At the end of computing, the vertices with the same colorID belong to one strongly-connected component.
- **transposeNeighbors**: stores neighbor IDs of v in the transpose graph of the input graph.

The algorithm is implemented as follows:

- **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.

- **Triming:** 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 vertexes are ignored.
- **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.
- **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 vertexes 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 there are still active vertices after this step, 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()));
}
}
}

```

```

    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<LongWri
table> msgs) 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().getScID());
}
}
/**
 * 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 = (My
Value)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 IOExcept
ion {
    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 Exception {
        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 +
            " seconds");
    }
}

```

9.6.6. Connected component

Two vertices are connected if a path exists between them. Undirected graph G is called a connected graph if every two vertices in the graph are connected. 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 outputs 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<Lo
            ngWritable> 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 IOExc
        eption {
        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 IOE
xception {
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 +
" seconds");
}

```



```
}
```

9.6.7. Topological sorting

For a directed edge (u,v) , all vertex sequences satisfying $u < v$ are called topological sequences. Topological sorting is an algorithm that is used to calculate the topological sequence of a directed graph.

The algorithm is implemented as follows:

- A vertex without incoming edges in the graph is found and output.
- The output vertex and all its outgoing edges are deleted.
- The preceding steps are repeated until all vertices are output.

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<LongWritable> 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 IOE
        xception {
        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 messa
        geToCombine) 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);
}
// Input format
// 0 1, 2
// 1 3
// 2 3
// 3 -1
// The first column is vertexid, and the second column is the destination vertexid of the v
// ertex. If the value is -1, the vertex does not have any outgoing edges.
// Output format
// 0 0
// 1 1
// 1 2
// 2 3
// The first column is the supstep value, in which the topological sequence is hidden. The
// second column is vertexid.
// TopologySortAggregator is used to determine if the graph has any loops.
// If the input graph has a loop, the iteration ends when the indegree of all active vertic
// es is not 0.
// You can use records in the input and output tables to determine if 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 +
    " seconds");
}
}

```

9.6.8. Linear regression

In statistics, linear regression is a statistical analysis method used to determine the dependency between two or more variables. Compared with the classification algorithm that predicts discrete data, the regression algorithm can predict continuous value-type data. 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 as follows:

- Initialize the weight vector to provide the descent speed and iterations (or iteration convergence condition).
- Calculate the least square error for each sample.
- Calculate the sum of the least square error, and update the weight based on the descent speed.
- 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);
```

```

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<NullWrita
        ble> messages) throws IOException {
        context.aggregate(getValue());
    }
}
public static class LinearRegressionVertexReader extends
GraphLoader<LongWritable, Tuple, NullWritable, NullWritable> {
    @Override
    public void load(LongWritable recordNum, WritableRecord record, MutationContext<LongWritabl
        e, 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:  $y(i) - h\theta(x(i))$  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).toString());
    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).get(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 * 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) lastTh
eta.get(j)).get(), 2);
if (
/*
* 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. Count triangles

This algorithm is used to calculate the number of triangles passing through each vertex. The algorithm is implemented as follows:

- Each vertex sends its ID to all outgoing neighbors.
- Each vertex stores information about incoming and outgoing neighbors, and sends this information to outgoing neighbors.
- Each vertex calculates the number of endpoint intersections for each edge, calculates the sum, and outputs the results to a table.
- The number of triangles is the sum of the output results in the table divided by 3.

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.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;
                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 +
" seconds");
}
}

```

9.6.10. GraphLoader

The following example describes how to compile a graph job program to load data of different types. It mainly covers how GraphLoader and VertexResolver are used together to build the graph.

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;
/**
 * A MaxCompute Graph job uses MaxCompute tables as the input. Assume that a job has two input tables, one storing vertices and the other storing edges.
 * The format of the table storing vertices is as follows:
 * +-----+
 * | VertexID | VertexValue |
 * +-----+
 * | id0 | 9 |
 * +-----+
 * | id1 | 7 |
 * +-----+
 * | id2 | 8 |
 * +-----+
 *
 * The format of the table storing edges is as follows:
 * +-----+
 * | VertexID | DestVertexID | EdgeValue |
 * +-----+
 * | id0 | id1 | 1 |
 * +-----+
 * | id0 | id2 | 2 |
 * +-----+
 * | id2 | id1 | 3 |
 * +-----+
 *
 * The two preceding tables show that id0 has two outgoing edges pointing to id1 and id2. id 2 has an outgoing edge pointing 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, while 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 in the load() method are combined to a vertex object, which is used as the returned value and added to the graph for participating in computation.
 */
public class VertexInputFormat {
    private final static String EDGE_TABLE = "edge.table";
    /**
     * Resolve a record to vertices and edges. Each record indicates a vertex or an edge based on its source.
     *
     * Enter a record to generate key-value pairs as you process com.aliyun.odps.mapreduce.Mapper#map. The keys are vertex IDs, and the values are vertices or edges written based on the context. These key-value pairs are summarized based on vertex IDs using LoadingVertexResolver.
     *
     * Note: Vertices or edges added here are requests sent based on the record content, and are not used in computation. Only vertices or edges added using VertexResolver participate in computation.
     */

```

```

public static class vertexinputLoader extends
GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
private boolean isEdgeData;
/**
 * Configure VertexInputLoader.
 *
 * @param conf
 * Indicate the configured parameters of a job. These parameters are configured in the MAIN
function of GraphJob, or set on the console.
 * @param workerId
 * Indicate the serial number of the operating worker. It starts from 0 and can be used to b
uild a unique vertex ID.
 * @param inputTableInfo
 * Indicate information about the input table loaded to the current worker. The information
can be used to determine the type of current input data (record format).
 */
@Override
public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableInfo());
}
/**
 * Based on the record content, resolve corresponding edges and send a request to add them t
o the graph.
 *
 * @param recordNum
 * Indicate the record serial number, which starts from 1 and is separately counted in each
worker.
 * @param record
 * Indicate the record in the input table. It contains three columns, indicating the first v
ertex, last vertex, and edge weight.
 * @param context
 * Indicate the context for adding resolved edges to the graph.
 */
@Override public void load(
LongWritable recordNum, WritableRecord record,
MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context) throws IOE
xception {
if (isEdgeData) {
/**
 * Data comes from the table that stores edges.
 *
 * 1. The first column indicates the first vertex ID.
 */
LongWritable sourceVertexID = (LongWritable) record.get(0);
/**
 * 2. The second column indicates the last vertex ID.
 */
LongWritable destinationVertexID = (LongWritable) record.get(1);
/**
 * 3. The third column indicates the edge weight.
 */
LongWritable edgeValue = (LongWritable) record.get(2);
/**
 * 4. Create an edge that consists of the last vertex ID and edge weight.
 */

```

```

    /
    Edge<LongWritable, LongWritable> edge = new Edge<LongWritable, LongWritable>( destinationVertexID, edgeValue);
    /**
    * 5. Send a request to add an edge to the first vertex.
    */
    context.addEdgeRequest(sourceVertexID, edge);
    /**
    * 6. If each record indicates a bidirectional edge, repeat steps 4 and 5. Edge<LongWritable
    , > LongWritable> edge2 = new
    * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
    * context.addEdgeRequest(destinationVertexID, edge2);
    */
    } else {
    /**
    * Data comes from the table that stores vertices.
    *
    * 1. The first column indicates the vertex ID.
    */
    LongWritable vertexID = (LongWritable) record.get(0);
    /**
    * 2. The second column indicates the vertex value.
    */
    LongWritable vertexValue = (LongWritable) record.get(1);
    /**
    * 3. Create a vertex that consists of the vertex ID and vertex value.
    */
    MyVertex vertex = new MyVertex();
    /**
    * 4. Initialize the vertex.
    */
    vertex.setId(vertexID); vertex.setValue(vertexValue);
    /**
    * 5. Send a request for adding a vertex.
    */
    context.addVertexRequest(vertex);
    }
    }
    }
    /**
    * Summarize key-value pairs generated using GraphLoader::load(LongWritable, Record, > MutationContext), which is similar to
    * Reduce in com.aliyun.odps.mapreduce.Reducer. For the unique vertex > ID, all actions such
    as
    * adding or deleting 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 IDs requested to be generated using the load() method are called.
    */
    public static class LoadingResolver extends
    VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
    /**
    * Process a request for adding/deleting 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 vertex objects with the same ID are requested to be added, such
vertex objects are collected to the returned list.
* VertexChanges::getAddedEdgeList() corresponds to
* MutationContext::addEdgeRequest(WritableComparable, Edge)
* If edge objects with the same first vertex ID are requested to be added, such edge object
s are collected to the returned list.
* VertexChanges::getRemovedVertexCount() corresponds to
* MutationContext::removeVertexRequest(WritableComparable)
* If vertices with the same ID are requested to be deleted, the number of total deletion re
quests is returned.
* VertexChanges#getRemovedEdgeList() corresponds to
* MutationContext#removeEdgeRequest(WritableComparable, WritableComparable)
* If edge objects with the same first vertex ID are requested to be deleted, such edge obje
cts are collected to the returned list.
*
* By processing ID changes, you can state whether the ID participates in computation using
the returned value. If the returned vertex is not NULL,
* the ID participates in subsequent computation. If the returned vertex is NULL, the ID doe
s not participate in subsequent computation.
*
* @param vertexId
* Indicate the ID of the vertex to be added, or the ID of the first vertex of the edge to b
e added.
* @param vertex
* Indicate an existing vertex object. Its value is always NULL in the data loading phase.
* @param vertexChanges
* Indicate the set of vertices or edges to be added/deleted for the ID.
* @param hasMessages
* Indicate whether the ID has any input messages. Its value is always false in the data loa
ding phase.
**/
@Override
public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve( LongWritable
vertexId,
Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex, VertexChanges<LongWr
itable, LongWritable, LongWritable, LongWritable> vertexChanges, boolean hasMessages) throw
s IOException {
/**
* 1. Obtain the vertex object for computation.
**/
MyVertex computeVertex = null;
if (vertexChanges.getAddedVertexList() == null
|| vertexChanges.getAddedVertexList().isEmpty()) { computeVertex = new MyVertex(); computeV
ertex.setId(vertexId);
} else {
/**
* Each record indicates a unique vertex in the table that stores 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, perform deduplication based on the algorithm needs.
**/
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 computation.
**/
return computeVertex;
}
}
/**
* Determine actions of the vertex that participates in computation.
*
**/
public static class MyVertex extends
Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {
/**
* Write the vertex edge to the result table based on the format of the input table. Ensure that the format and data of the input and output tables are the same.
*
* @param context
* Indicate the runtime context.
* @param messages
* Indicate the input message.
**/
@Override
public void compute(
ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context, Iterable<LongWritable> messages) throws IOException {
/**
* Write the vertex ID and value to the result table that stores vertices.
**/
context.write("vertex", getId(), getValue());
/**
* Write the vertex edge to the result table that stores edges.
**/
if (hasEdges()) {
    for (Edge<LongWritable, LongWritable> edge : getEdges()) { context.write("edge", getId(), edge.getDestVertexId(), edge.getValue());
    }
}
/**
* Perform one round of 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>");
    }
    /**
     * GraphJob is used to configure Graph jobs.
     */
    GraphJob job = new GraphJob();
    /**
     * 1. Specify input graph data and the table that stores 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. Similar to Map, the g
     * enerated key is the vertex ID, and the value is the edge.
     */
    job.setGraphLoaderClass(VertexInputLoader.class);
    /**
     * 3. Specify the data loading phase, and generate the vertex that participates in computati
     * on. Similar to Reduce, edges generated by Map are combined to a vertex.
     */
    job.setLoadingVertexResolverClass>LoadingResolver.class);
    /**
     * 4. Specify actions of the vertex that participates in computation. The vertex.compute() m
     * ethod is used for each round of iteration.
     */
    job.setVertexClass(MyVertex.class);
    /**
     * 5. Specify the output table of the Graph job, and write the computation result to the res
     * ult table.
     */
    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

MaxCompute provides Java SDK with a variety of APIs to support development on MaxCompute.

For more information about the APIs, see *MaxCompute Developer Guide*.

11.PyODPS

11.1. Overview

Although you can implement most MaxCompute development by executing SQL statements, you need to use Python in complex business scenarios and custom UDF scenarios.

PyODPS is the Python SDK of MaxCompute. It provides simple and convenient Python programming interfaces, basic operations on MaxCompute objects, and the DataFrame framework. It allows you to easily analyze data on MaxCompute.

11.2. Quick start

This topic describes how to use a PyODPS node in DataWorks for development. The following procedure is for reference only.

1. Create a PyODPS node.

- i. Create a business flow.

Right-click **Business Flow** below **Data Analytics** and choose **Create Workflow** from the shortcut menu.

- ii. Create a node.

Right-click **Data Analytics** and choose **Create Data Analytics Node > PyODPS**.

2. Edit the PyODPS node.

- i. Write the program code.

The following code is for reference only:

```
import time                # Similar to Java, module import is required before a
                             n additional SDK is called.
import datetime            # In this example, only the print function is used.
import base64
import hashlib
import httplib
import json
import sys
import csv
from odps import ODPS      # To call an SDK related to MaxCompute, you must impo
                             rt this module.
def main():
    print("Hello World")    # Provide the output in the log file.
if __name__ == "__main__": # The main entry of the program.
    main()
```

- ii. Run the code.

After you edit the code, click the Run icon. You can view the running results in the **Runtime Log** section.

The first code snippet is completed. The main entry of the code is determined based on `if __name__ == "__main__"`. This statement takes effect only when the preceding script is run directly (name = main). It does not take effect when it is referenced as a module by other code files (name = Python file name).

11.3. Installation instructions

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 instructions](#). If you want to speed up the download, we recommend that you use [Alibaba Cloud images](#).

Prerequisites

Before you install PyODPS, make sure that the following requirements are met:

- 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 speed up Tunnel upload:

- Greenlet. Recommended version: 0.4.10 or later.
- Cython. Recommended version: 0.19.0 or later.

Installation commands for reference:

```
pip install greenlet>=0.4.10 # Optional. It accelerates Tunnel upload.
pip install cython>=0.19.0 # Optional. It is not recommended 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 speed up Tunnel upload.

Installation procedure

Run the following command to install PyODPS:

```
pip install pyodps
```

Run the following command to check whether the installation is complete:

```
python -c "from odps import ODPS"
```

If the Python version is not the default, you can 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 # Replace the version of Setuptools with the actual version.
```

11.4. Platform instructions

11.4.1. 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 supports scheduling and execution. You do not need to manually create a MaxCompute object. To migrate from a platform to a locally deployed PyODPS environment, see the instructions in the next topic.

11.4.2. Use local PyODPS

If you need 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.

You must install PyODPS first. For information about how to install it, see [Installation instructions](#).

After you have installed PyODPS, you must manually create the MaxCompute object that was previously created on the platform. Then, execute the following statement on the platform to generate a statement template required by the MaxCompute object and manually modify the code:

```
print("\nfrom odps import ODPS\no = ODPS(%r, '<access-key>', %r, '<endpoint>')\n" % (o.access_id, o.project))
```

 **Note** You need to replace `access-key` and `endpoint` with valid values. For information about how to obtain the values, see the corresponding topics in *MaxCompute Developer Guide*.

Then, place the modified code at the beginning of all code.

11.4.3. Use PyODPS in DataWorks

Create a workflow node

Set Node Type to PYODPS.

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

For more information, see [SQL](#).

 **Note** 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. After `InstanceTunnel` is enabled, you can execute `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 also add `tunnel=True` and `limit=False` to `open_reader` to enable InstanceTunnel and 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 incorrect.

DataFrame

- Execute DataFrame.

To execute DataFrame 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 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`. By default, this parameter is set to `True` in DataWorks. The system displays details such as the LogView URL while running.

Obtain scheduling parameters.

Different from SQL nodes in DataWorks, a PyODPS node does not replace strings such as `${param_name}` in the code. Instead, it adds a dict parameter named `args` to the global variable. You can obtain the scheduling parameters in dict. This way, the Python code is not affected. For example, if you set `ds=${yyyymmdd}` under **Schedule > Parameter** in DataWorks, you can run the following commands 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

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.
- User defined functions (UDFs) of DataFrame 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 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 need to use the `try-finally` structure to implement related functions.

Limits on usage

- The Python version of a PyODPS node is 2.7.
- Locally processed data obtained by a PyODPS node cannot exceed 50 MB, and the memory space occupied by the node cannot exceed 1 GB. Otherwise, the system stops tasks in the node. Do not write unnecessary Python data processing code in PyODPS tasks.
- Writing and debugging code in DataWorks is inefficient. We recommend that you install an IDE locally to write code.
- To prevent excess pressure on the gateway of DataWorks, the memory usage and CPU utilization are limited when PyODPS is used in DataWorks. The limits are managed by DataWorks. If the system displays **Got killed**, the memory usage exceeds the limit and the system stops the related processes. Therefore, we do not recommend you perform local data operations. However, the limits on the memory usage and CPU utilization do not apply to SQL and DataFrame tasks (except `to_pandas`) that are initiated by PyODPS.

11.5. Basic operations

11.5.1. Overview

PyODPS provides basic operations for MaxCompute objects. You can use Python-compliant programming methods to perform operations on MaxCompute.

11.5.2. Projects

You can use the `get_project` method of a MaxCompute object to obtain a project.

```
project = o.get_project('my_project') # Obtain a specific project.
project = o.get_project()             # Obtain the default project.
```

If the `my_project` parameter is not specified, the default project is obtained.

You can use the `exist_project` method to check whether a project exists.

11.5.3. Tables

Basic operations

You can call `list_tables` for a MaxCompute object to list all tables in a project.

```
for table in o.list_tables():  
    # Process each table.
```

You can call `exist_table` to check whether a table exists and call `get_table` to obtain the table.

```
t = o.get_table('dual')
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
'Dual 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('dual', project='other_project')
```

Create a table schema

You can initialize a table in two ways:

- Initialize a table based on table columns and available 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 types of non-partition fields.
[bigint, double]

```

- Initialize a table by calling `Schema.from_lists`. This method is easier to call, but you cannot directly set the comments of 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 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) #The table is created o
nly if no table with the same name exists.
table = o.create_table('my_new_table', schema, lifecycle=7) # Set the lifecycle.

```

You can also use a comma-connected string in the "field name, field type" format to create a table. This method is easier.

```

table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
# To create a partitioned table, you can pass in (table field list, partition field list).
table = o.create_table('my_new_table', ('num bigint, num2 double', 'pt string'), if_not_exi
sts=True)

```

By default, when you create a table, you can only use 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 can 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), bo
dy string>')

```

Synchronize table updates

After another program updates a table, such as its schema, you can call the `reload` method to synchronize the update.

```
table.reload()
```

Record

A record refers to a single row in a table. You can call `new_record` for the table object to create a record.

```
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 a 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 obtain table data in three ways:

- Call `head` to retrieve up to the first 10,000 data records in a table. Example:

```
t = odps.get_table('dual')
for record in t.head(3):
    print(record[0]) # Obtain the value at position 0.
    print(record['c_double_a']) # Obtain a value based on a field.
    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.
```

- Use `open_reader` for a table object to open a reader and read the table data. You can open the reader with or without a `WITH` clause.

```
# 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 this line multiple times until all records are read. The number of records is specified by count. You can change it 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 this line multiple times until all records are read. The number of records is specified by count. You can change it to parallel-operation code.
    # Process a record.
```

- Use the `read_table` method for a MaxCompute object to obtain table data. Example:

```
for record in o.read_table('test_table', partition='pt=test'):
    # Process a record.
```

Write data to a table.

Similar to `open_reader`, you can use `open_writer` for a table object to open a writer and write data to a table. Example:

```
# Open the writer with a WITH clause:
with t.open_writer(partition='pt=test') as writer:
    records = [[111, 'aaa', True], # A list can be used.
               [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)
with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer: # Two blocks are enabled
    .
    writer.write(0, gen_records(block=0))
    writer.write(1, gen_records(block=1)) # The two write operations can be performed in parallel based on the multithreading technique. Each block is independent.
# Open the writer without a WITH clause:
writer = t.open_writer(partition='pt=test', blocks=[0, 1])
writer.write(0, gen_records(block=0))
writer.write(1, gen_records(block=1))
writer.close() # You must close the writer. Otherwise, the written data may be incomplete
.
```

If the specified partition does not exist, use the `create_partition` parameter 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 used.
               [222, 'bbb', False],
               [333, 'ccc', True],
               [444, 'Chinese', False]]
    writer.write(records) # records can be iterable objects.
```

You can also use the `write_table` method for a MaxCompute object to write data. Example:

```
records = [[111, 'aaa', True],          # A list can be used.
           [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 `write_table`, MaxCompute generates a file on the server. This operation is time-consuming. If too many files are generated, the efficiency of subsequent queries will be reduced. Therefore, we recommend that you write multiple records at a time or provide a Generator object when you use the `write_table` method.
- When you use `write_table`, new data will be appended to existing data. PyODPS does not provide options to overwrite existing data. You need to manually remove the data that you want to overwrite. For a non-partitioned table, you must call `table.truncate()`. For a partitioned table, you need to delete partitions first.

Delete a table

```
o.delete_table('my_table_name', if_exists=True) # Delete a table if the table exists.
t.drop() # The drop function can be directly executed if a table object exists.
```

Create a DataFrame object

PyODPS provides a DataFrame framework, which allows you to conveniently query and manage MaxCompute data. You can use `to_df` to convert a table to a DataFrame object.

```
table = o.get_table('my_table_name')
df = table.to_df()
```

Table partitions

- Basic operations

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 subpartitions.
```

Check whether a specific partition exists:

```
table.exist_partition('pt=test,sub=2019')
```

Obtain a specific partition:

```
partition = table.get_partition('pt=test')
print(partition.creation_time)
2019-09-18 22:22:27
partition.size
0
```

- Create a partition.

```
t.create_partition('pt=test', if_not_exists=True) # Create the partition if it does not exist.
```

- Delete a partition.

```
t.delete_partition('pt=test', if_exists=True) # Delete the partition if it exists.
partition.drop() # Use the drop method to delete the partition object if it exists.
```

Data upload and download channels

MaxCompute Tunnel is the data channel of MaxCompute. You can use it to upload data to or download data from MaxCompute.

 **Note** We recommend that you use the write and read interfaces of tables instead of the Tunnel interface. In a Cython environment, PyODPS compiles C programs during installation to accelerate the Tunnel upload and download.

- Upload example:

```
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])
```

- Download example:

```
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 currently does not allow you to upload external tables.

11.5.4. SQL

PyODPS supports MaxCompute SQL queries and provides methods to read execution results.

You can use the `execute_sql` and `run_sql` methods to create task instances.

 **Note** The commands that are executable in the MaxCompute console may not be executed as SQL statements in MaxCompute. Use other methods to execute non-DDL/DML statements. For example, use the `run_security_query` method to execute GRANT/REVOKE statements, and use the `run_xflow` or `execute_xflow` method to execute PAI commands.

Execute SQL statements

```
o.execute_sql('select * from dual') # Run the SQL statement in synchronous mode. Blocking
continues until execution of the SQL statement is completed.
instance = o.run_sql('select * from dual') # Run the SQL statement in asynchronous mode.
print(instance.get_logview_address()) # Obtain the LogView address.
instance.wait_for_success() # Blocking continues until execution of the SQL statement is c
ompleted.
```

Configure runtime parameters

You can configure runtime parameters by setting the `hints` parameter. The type of this parameter is DICT.

```
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 base
d on global settings.
```

Obtain SQL query results

You can directly call the `open_reader` method to obtain SQL query results. In the following example, structured data is returned.

```
with o.execute_sql('select * from dual').open_reader() as reader:
    for record in reader:
        # Process each record.
```

When 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 `options.tunnel.use_instance_tunnel` is set to `True` when you use `open_reader`, PyODPS calls Instance Tunnel by default. Otherwise, PyODPS calls the old Result interface. However, if you are using an earlier version of MaxCompute or an error occurs when PyODPS calls Instance Tunnel, 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 Instance Tunnel does not meet your expectation, set this option to `False`. When you call `open_reader`, you can specify a 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. Generally, 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 SQL query results 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 `open_reader` to call Instance Tunnel 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 alias

During runtime, if the resource referenced by a UDF changes dynamically, you can use the old resource name as the alias of the current resource. This way, you do not need to delete the UDF or create a 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')
# Use the name of the resource whose content is 1 as the alias of the resource whose content is 2. You do not 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, when an SQL statement is executed, biz_id must be submitted. 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')
```


11.5.5. Task instances

Tasks, such as SQL tasks, are the basic computing unit of MaxCompute.

Basic operations

Tasks are implemented as MaxCompute instances. You can call `list_instances` to obtain all instances in a project, `exist_instance` to check whether an instance exists, and `get_instance` to obtain an instance.

```
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 address of a task

You can call `get_logview_address` to obtain the LogView address of an SQL task.

```
# Obtain the LogView address based on an existing instance object.
instance = o.run_sql('desc pyodps_iris')
print(instance.get_logview_address())
# Obtain the LogView address based on an instance ID.
instance = o.get_instance('2016042605520945g9k5pvyi2')
print(instance.get_logview_address())
```

For an XFlow task, you must enumerate its subtasks and then obtain the LogView address of each subtask.

```
instance = o.run_xflow('AppendID', 'algo_public', {'inputTableName': 'input_table', 'output
TableName': '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 a task instance

An instance can be in the **Running**, **Suspended**, or **Terminated** state. You can obtain the status from the status attribute. You can call the `is_terminated` method to check whether the execution of the current instance is completed 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 instance execution is completed. The `wait_for_success` method functions similarly. The difference is that if the execution fails, an exception is reported.

Subtask operations

A running instance may contain one or more subtasks, which are called Tasks. Note that Task here does not refer to the basic computing unit of MaxCompute.

You can call the `get_task_names` method to obtain the names of all Tasks.

```
instance.get_task_names()
['SQLDropTableTask']
```

After obtaining a Task name, you can call `get_task_result` to obtain the execution result of the Task. You can also call `get_task_results` to obtain the execution result of each Task 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 Task when the task 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_Stgl_job0:0/1/1[100%]
```

11.5.6. Resources

PyODPS mainly supports file and table resources.

Resources commonly apply to UDFs and MapReduce on MaxCompute.

You can call `list_resources` to list all resources, `exist_resource` to check whether a resource exists, and `delete_resource` to delete a resource. You can also call the `drop` method to delete a resource.

File resources

Files of basic file types and the .py, .jar, and .archive types are supported.

- Create a file resource

You can create a file resource by specifying a resource name, a file type, and a file-like or string object. Example:

```
resource = odps.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file')) # Use a file-like object.
resource = odps.create_resource('test_py_resource', 'py', file_obj='import this') # Use a string object.
```

- Read and modify a file resource

You can call the `open` method for a file resource or call `open_resource` 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. Example:

```
with resource.open('r') as fp: # Open the file in read mode.
    content = fp.read() # Read all content.
    fp.seek(0) # Return to the beginning of the resource.
    lines = fp.readlines() # Read multiple lines.
    fp.write('Hello World') # Error. Data cannot be written into a file in read mode.
with odps.open_resource('test_file_resource', mode='r+') as fp: # Open the file in read/write mode.
    fp.read()
    fp.tell() # Current position
    fp.seek(10)
    fp.truncate() # Truncate the file to the specified length.
    fp.writelines(['Hello\n', 'World\n']) # Write multiple lines into the file.
    fp.write('Hello World')
    fp.flush() # Manually calling the method will submit the update to MaxCompute.
```

PyODPS supports the following opening modes:

- **r**: read mode. The file can be opened but data cannot be written into it.
- **w**: write mode. Data can be written into the file, but data in it cannot be read. Note that the file content is cleared first if the file is opened in write mode.
- **a**: append mode. Data can be added to the end of the file.
- **r+**: read/write mode. You can read and write data from and to the file.
- **w+**: This mode is similar to **r+**, but the file content is cleared first.
- **a+**: This mode is similar to **r+**, but data can only be written 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.

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 and write data

```
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)
```

11.5.7. Functions

You can create UDFs and use them in MaxCompute SQL.

Basic operations

You can call `list_functions` of a MaxCompute object to obtain all functions in the project. You can call `exist_function` to check whether a function exists and call `get_function` to obtain a function.

Create a function

```
# Reference resources 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 resources 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

```
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.

```
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.
```

11.6. DataFrame

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, so that the same code can be executed on pandas.

This topic describes how to create and perform operations on DataFrame objects and how to use DataFrame to process data. For the complete DataFrame document, see [DataFrame](#).

DataFrame object operations

The following example demonstrates how to create a DataFrame object. It is for reference only.

 **Note** The data used in the example is downloaded from [movielens 100K](#). You can download the data as needed.

Assume that the following three tables already exist: pyodps_ml_100k_movies (movie-related data), pyodps_ml_100k_users (user-related data), and pyodps_ml_100k_ratings (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 only need to import 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 DataFrame and the types of these 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 obtain the first N data records for quick data preview.
Example:

```
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. You can add a filter on the fields if you do not want to view all of them.

o Example:

```
users[['user_id', 'age']].head(5)
```

	user_id	age
0	1	24
1	2	53
2	3	23
3	4	24
4	5	33

o You can also 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

o When you exclude some fields, you may want to obtain new columns through 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. 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, 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. Note that the number of records returned by this method is limited by `options.df.odps.sort.limit`.

```
users.occupation.value_counts()[:10]
```

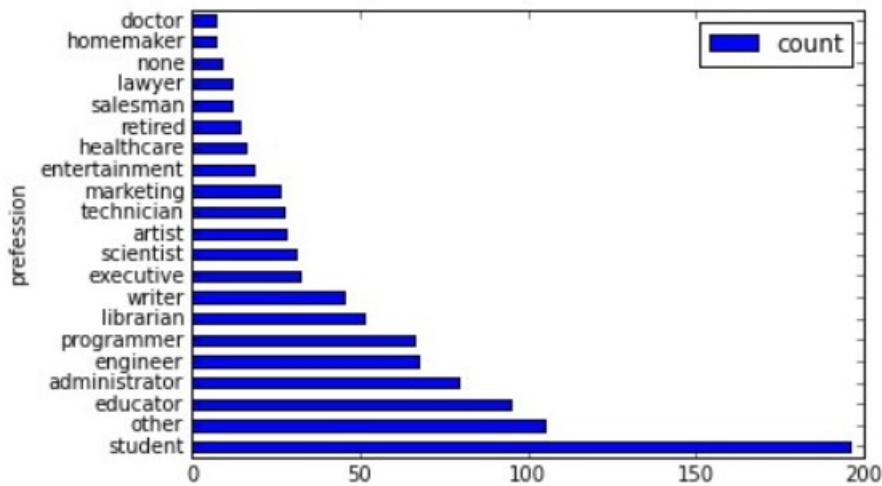
	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32
9	scientist	31

8. Show data in a more intuitive graph.

```
%matplotlib inline
```

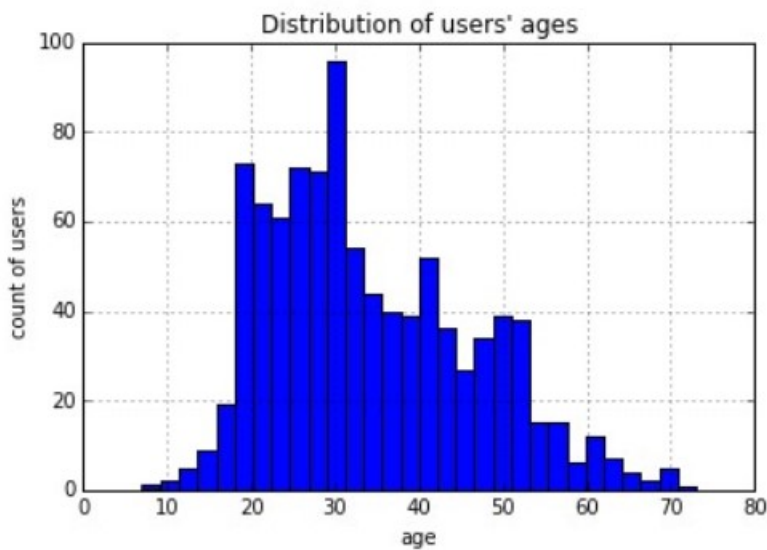
9. Use a horizontal column chart to visualize data.

```
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. It is for reference only.

 **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 **By Location** and click **Import**.

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, typ
es='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 col
umn extends horizontally.
print iris.apply(handle,axis=1).head(5)
```

11.7. User experience enhancement

11.7.1. Command line

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 have configured an account, you do not need to enter the account information again upon the next login.

3. You can then call the enter method to create a Room object in any 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 object is not automatically updated when you change the setup of the room. You need to call `enter()` again to retrieve the new Room object.

After you have configured an account and called 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()
```

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_sqldtask_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.

```
room.drop('stored-table')
room.display()
default name      desc
iris              Iris dataset
```

- You can call `teardown()` to delete a room. When no parameter is specified, the default room is deleted.

```
teardown()
```

11.7.2. IPython

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
```

Result:

```
<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')
```

Result:

```
odps.Table
name: odps_test_sqldtask_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
```

- Run the following command to display the stored objects as a table:

```
%stores
```

Result:

default name	desc
iris	Iris dataset

Object name completion

PyODPS enhances the code completion feature that is provided by IPython. When you write statements 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.

- 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 can also complete the names of objects in different projects. The syntax is as follows:

```
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 a 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[*]:
```

	sepalength	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 sepalength < 5
....: limit 5
....:
|=====| 1 / 1 (100.00%) 15s
Out[*]:
```

	sepalength	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 a 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 the following command to upload pandas DataFrame objects to MaxCompute tables:

You only need 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
```

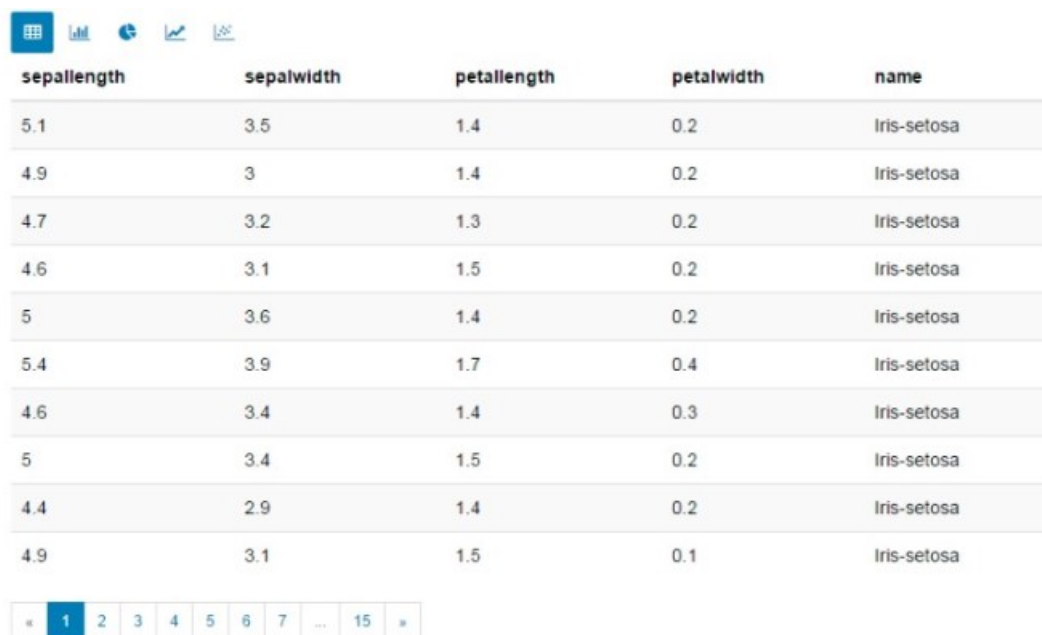
11.7.3. Jupyter Notebook

PyODPS enhances the result exploration and progress display features of Jupyter Notebook.

Result exploration

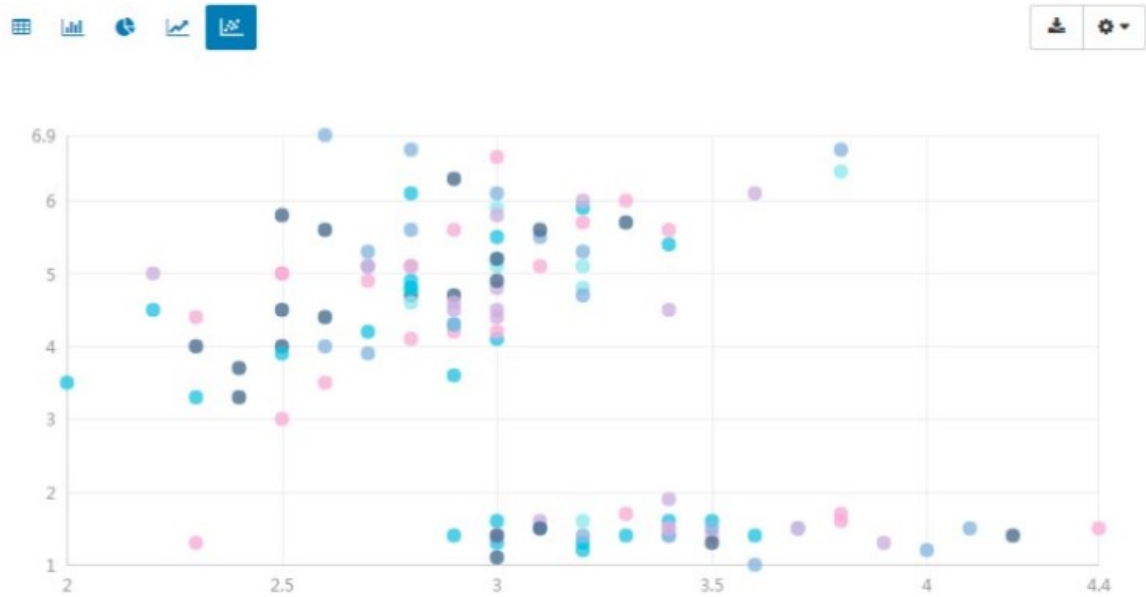
PyODPS provides a data exploration feature in Jupyter Notebook for SQLCell and DataFrame. You can use interactive data exploration tools to browse local data and create graphs.

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 or the Previous or Next button to browse the data.

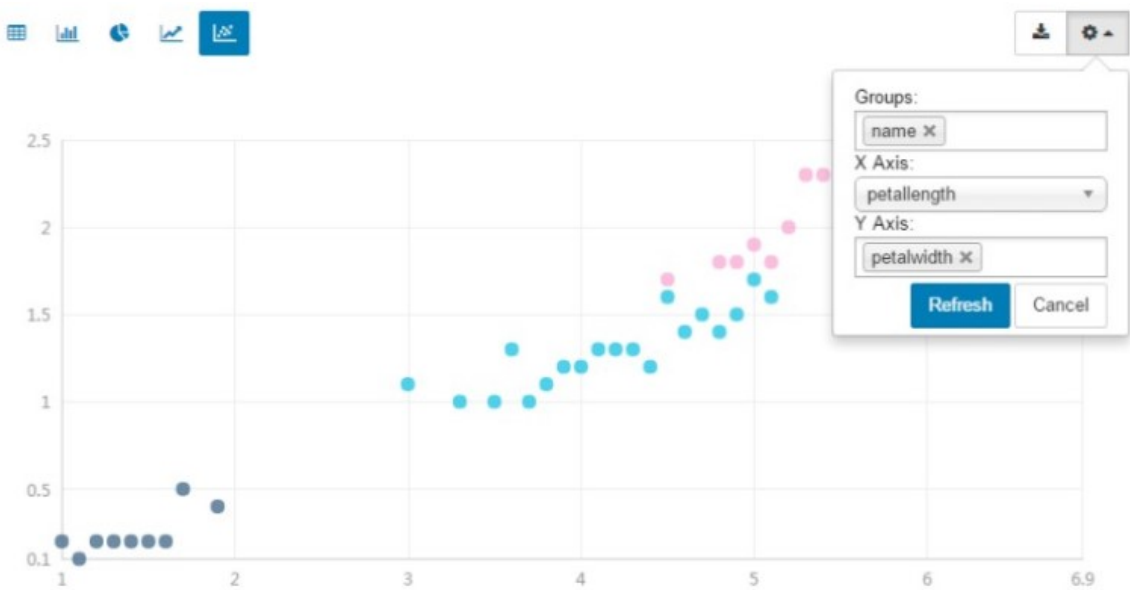


sepalwidth	sepalwidth	petalwidth	petalwidth	name
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa

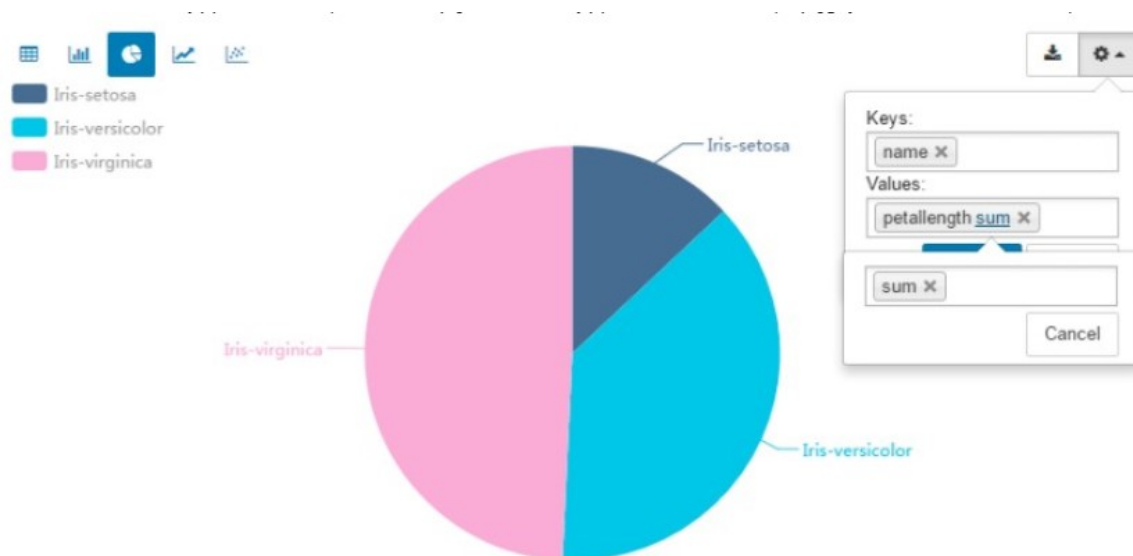
2. You can select other display modes on the top of the table to display the result in a column chart, pie chart, line chart, or scatter chart. The following figure shows a scatter chart created based on the default fields, which are the first three fields.



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** setting displays the data in a manner that is easy to understand.



For bar 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 value field name to select another function. For line charts, ensure that the values on the x-axis are not null. Otherwise, the graph may not display correctly.



4. After you finish drawing the graph, click the **Download** icon to save it.

Note Before you use this feature, ensure that you have installed pandas and ipywidgets.

Progress display

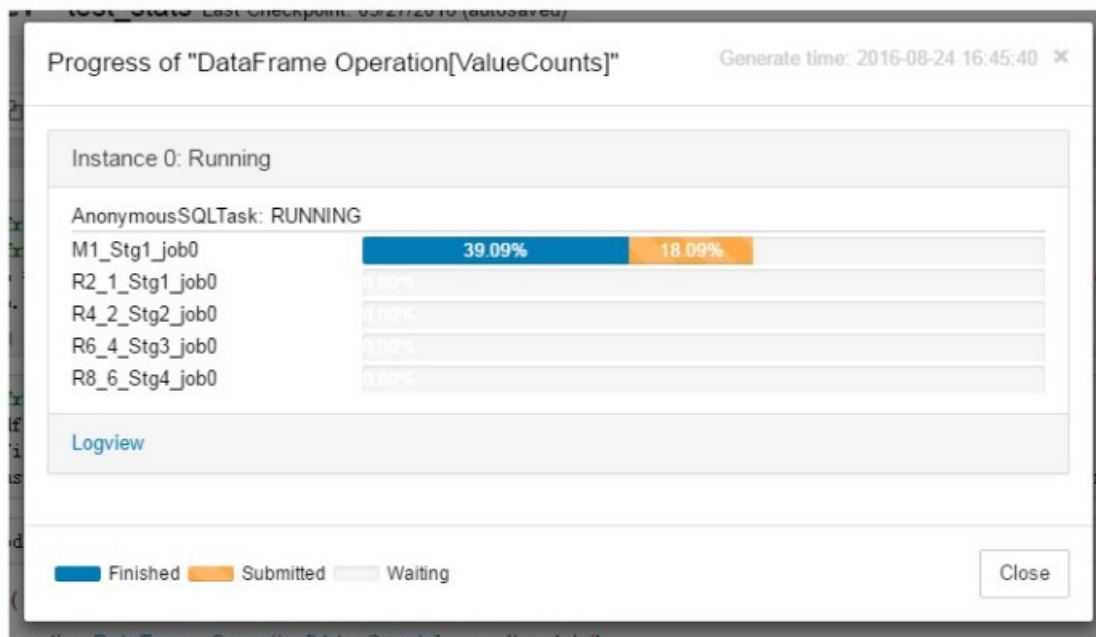
The execution of large jobs takes extended periods of time. PyODPS provides progress bars to show the execution progress. When DataFrame jobs, machine learning jobs, or SQL statements that start with `%sql` are executed in Jupyter Notebook, a list of these jobs and their overall progress are displayed.

```
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()
```

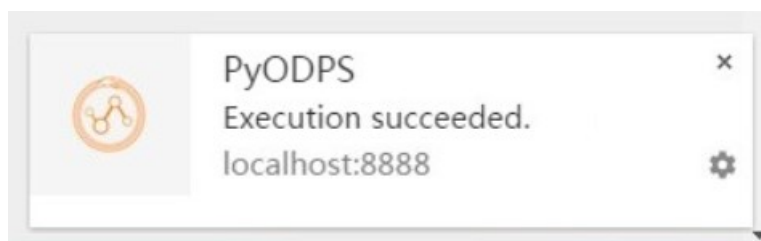
✕ (50.00%)

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.



11.8. Configuration

PyODPS provides a series of configuration options. You can obtain them by using `odps.options`.

Example:

```
from odps import options
# Set the lifecycle option, which specifies 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 DataFrame on 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 configuration

Option	Description	Default value
end_point	The endpoint of MaxCompute.	None

Option	Description	Default value
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 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 (unit: seconds).	5
read_timeout	The read timeout period (unit: seconds).	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
notebook_repr_widget	Specifies whether to use interactive graphs.	True
sql.settings	MaxCompute SQL runs global hints.	None

Option	Description	Default value
sql.use_odps2_extension	Specifies whether to enable MaxCompute 2.0 language extension.	False

Data upload and download configuration

Option	Description	Default value
tunnel.endpoint	The endpoint of Tunnel.	None
tunnel.use_instance_tunnel	Specifies whether to use Instance Tunnel to obtain execution results.	True
tunnel.limit_instance_tunnel	Specifies whether to limit the number of data records obtained by using Instance Tunnel.	None
tunnel.string_as_binary	Specifies whether to use bytes instead of unicode for data of the STRING type.	False

DataFrame configuration

Option	Description	Default value
interactive	Specifies whether DataFrame is used in an interactive environment.	Depends on the detection value.
df.analyze	Specifies whether to enable non-MaxCompute built-in functions.	True
df.optimize	Specifies whether to enable full DataFrame optimization.	True
df.optimizes.pp	Specifies whether to enable DataFrame predicate push 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
df.quote	Specifies whether to use `` to mark fields and table names in the backend of MaxCompute SQL.	True

Option	Description	Default value
df.libraries	The third-party library (resource name) that is used by DataFrame.	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 added during a sort operation of DataFrame.	10000

Machine learning configuration

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 model PMML.	False
ml.model_volume	The name of the volume used by ModelTransfer.	pyodps_volume

11.9. API overview

Links to PyODPS API documentation are provided below. You can see the parameter description and example of each function:

- [Definitions](#)
- [DataFrame Reference](#)

11.10. FAQ

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 incorrect 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 currently 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, or you can store DataFrame as a table and use `read_table` or Tunnel to read data.

12.Java sandbox limits

MaxCompute MapReduce and UDF programs in distributed environments are run in Java sandboxes. The main programs of MapReduce are not subject to these limits.

The limits include:

- Local files cannot be accessed directly, and can only be accessed through APIs provided by MaxCompute MapReduce or Graph. You can access the resources specified by the resources option, such as files, JAR packages, and resource tables. You can use System.out and System.err to export logs and run the log command on the MaxCompute console to view log information.
- Direct access to the distributed file system is not allowed. You can only use MaxCompute MapReduce to Graph to access records of tables.
- JNI calls are not allowed.
- Java threads cannot be created, and Linux commands cannot be executed by sub-threads.
- Network access, including the operation of getting the local IP address, is prohibited.
- Java reflection restriction: The suppressAccessChecks permission is prohibited, so you cannot set a private attribute or method accessible to read private attributes or call private methods.

Specifically, an access denied error is returned when you perform any of the preceding operations.

Methods for accessing local files:

java.io.File:

```
public boolean delete()
public void deleteOnExit()
public boolean exists()
public boolean canRead()
public boolean isFile()
public boolean isDirectory()
public boolean isHidden()
public long lastModified()
public long length()
public String[] list()
public String[] list(FilenameFilter filter)
public File[] listFiles()
public File[] listFiles(FilenameFilter filter)
public File[] listFiles(FileFilter filter)
public boolean canWrite()
public boolean createNewFile()
public static File createTempFile(String prefix, String suffix)
public static File createTempFile(String prefix, String suffix,File directory)
public boolean mkdir()
public boolean mkdirs()
public boolean renameTo(File dest)
public boolean setLastModified(long time)
public boolean setReadOnly()
java.io.RandomAccessFile:
RandomAccessFile(String name, String mode)
RandomAccessFile(File file, String mode)
java.io.FileInputStream:
FileInputStream(FileDescriptor fdObj)
FileInputStream(String name) FileInputStream(File file)
```



```

PrintStream(PrintStream out, PrintStream err,
java.io.FileOutputStream:
FileOutputStream(FileDescriptor fdObj)
FileOutputStream(File file)
FileOutputStream(String name)
FileOutputStream(String name, boolean append)
java.lang.Class:
public ProtectionDomain getProtectionDomain()
java.lang.ClassLoader:
ClassLoader ()
ClassLoader(ClassLoader parent)
java.lang.Runtime:
public Process exec(String command)
public Process exec(String command, String envp[])
public Process exec(String cmdarray[])
public Process exec(String cmdarray[], String envp[])
public void exit(int status)
public static void runFinalizersOnExit(boolean value)
public void addShutdownHook(Thread hook)
public boolean removeShutdownHook(Thread hook)
public void load(String lib)
public void loadLibrary(String lib)
java.lang.System:
public static void exit(int status)
public static void runFinalizersOnExit(boolean value)
public static void load(String filename)
public static void loadLibrary( String libname)
public static Properties getProperties()
public static void setProperties(Properties props)
public static String getProperty(String key)
// Only some keys are accessible.
public static String getProperty(String key, String def)
// Only some keys are accessible.
public static String setProperty(String key, String value)
public static void setIn(InputStream in)
public static void setOut(PrintStream out)
public static void setErr(PrintStream err)
public static synchronized void setSecurityManager(SecurityManager s)

```

List of keys allowed by System.getProperty:

```

java.version
java.vendor
java.vendor.url
java.class.version
os.name os.version
os.arch
file.separator
path.separator
line.separator
java.specification.version
java.specification.vendor
java.specification.name
java.vm.specification.version
java.vm.specification.vendor

```

```
java.vm.specification.name
java.vm.version
java.vm.vendor
java.vm.name
file.encoding
user.timezone
java. lang. Thread:
Thread ()
Thread(Runnable target)
Thread(String name)
Thread(Runnable target, String name)
Thread(ThreadGroup group, ...)
public final void checkAccess()
public void interrupt()
public final void suspend()
public final void resume()
public final void setPriority (int newPriority)
public final void setName(String name)
public final void setDaemon(boolean on)
public final void stop()
public final synchronized void stop(Throwable obj)
public static int enumerate(Thread tarray[])
public void setContextClassLoader(ClassLoader cl)
java. lang. ThreadGroup:
ThreadGroup (String name)
ThreadGroup(ThreadGroup parent, String name)
public final void checkAccess()
public int enumerate(Thread list[])
public int enumerate(Thread list[], boolean recurse)
public int enumerate(ThreadGroup list[])
public int enumerate(ThreadGroup list[], boolean recurse)
public final ThreadGroup getParent()
public final void setDaemon(boolean daemon)
public final void setMaxPriority(int pri)
public final void suspend()
public final void resume()
public final void destroy()
public final void interrupt()
public final void stop()
java. lang. reflect. AccessibleObject:
public static void setAccessible (...)
public void setAccessible (...)
java.net. InetAddress:
public String getHostName ()
public static InetAddress[] getAllByName(String host)
public static InetAddress getLocalHost()
java.net.DatagramSocket:
public InetAddress getLocalAddress()
java.net.Socket:
Socket(...)
java.net.ServerSocket:
ServerSocket (...)
public Socket accept()
protected final void implAccept(Socket s)
```

```
public static synchronized void setSocketFactory(...)
public static synchronized void setSocketImplFactory(...)
java.net.DatagramSocket:
DatagramSocket (...)
public synchronized void receive(DatagramPacket p)
java.net.MulticastSocket:
MulticastSocket(...)
java.net.URL:
URL(...)
public static synchronized void setURLStreamHandlerFactory(...)
java.net.URLConnection
public static synchronized void setContentHandlerFactory(...)
public static void setFileNameMap(FileNameMap map)
java.net.HttpURLConnection:
public static void setFollowRedirects(boolean set)
java.net.URLClassLoader
URLClassLoader(...)
java.security.AccessControlContext:
public AccessControlContext(AccessControlContext acc, DomainCombiner combiner)
public DomainCombiner getDomainCombiner()
```

13. Volume lifecycle management

13.1. Overview

In the previous version of MaxCompute, a partition in a volume does not have a lifecycle and can exist forever. You have to manage the lifecycle of a volume. In some cases, user management of volume lifecycle 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 can solve this problem.

For more information about simple volume lifecycle management operations, see *Volume lifecycle operations*.

13.2. Volume lifecycle operations

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

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 provides a native Spark user experience 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 jobs and ensure high performance at a low cost. Spark can work with MaxCompute to create better and more efficient data processing solutions. Community Spark applications can run in **Spark on MaxCompute**.

Spark on MaxCompute has an independent data development node in DataWorks and supports data development in DataWorks.

14.2. Project resources

Before you use **Spark on MaxCompute**, you may need to pay attention to or download the following project resources:

- **Spark on MaxCompute** release package: Download the latest release package at [Spark on MaxCompute](#).
- **Spark on MaxCompute** plugin: This is an open-source program. Download the plugin at [Aliyun Cupid SDK](#).

After you prepare the preceding project resources, you need to complete environment configuration. Then, you can run related GitHub demos.

14.3. Environment settings

14.3.1. Decompress the Spark on MaxCompute release package

Download the latest version of the **Spark on MaxCompute** release package and decompress it. The structure of the decompressed folder is as follows:

```
.  
|-- R  
|-- RELEASE  
|-- __spark_libs__.zip  
|-- bin  
|-- conf  
|-- cupid  
|-- derby.log  
|-- examples  
|-- jars  
|-- logs  
|-- metastore_db  
|-- python  
|-- sbin  
|-- yarn
```

14.3.2. Set environment variables

Set required environment variables.

 **Note** 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

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 content is the default configuration in the file. You only need to enter your account information in the blanks.

```
# 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 how to use Spark on MaxCompute.

1. Download the Spark package from [Spark on MaxCompute](#) and decompress the 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. Set spark-defaults.conf.

```
cp $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-defaults.conf
```

Edit \$SPARK_HOME/conf/spark-defaults.conf by filling information in the blanks.

```
# 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-SNAPS
HOT-shaded.jar
```

If you see the following output, 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=20
180209075148695gbkp8e01&t=spark&id=application_1518162700322_1635034099&metaname=2018020907
5148695gbkp8e01&token=YkhjNXJWZ0dvdzVSXVFQWpCQWMra1RSZHVFPsXPRFBTX09CTzoxMzY1OTM3MTUwNzcyM
jEzLDElMTg0MjE5MzUseyJTdGF0ZWl1bnQiOlt7IkFjdGlvb2I6I6WyJvZHBzOlJlYWQiXSwiRWZmZWNOIjoiQWxs
b3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL29kcHNfc21va2VfdGVzdC9pbmN0YW5jZXMvMjAxODAyMDk
wNzUxNDg2OTVnYmtwOGUwMSJdfV0sIlZlcnNpb24iOiIxIn0=
  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-79a8-49f
7-931d-0533663b5d85

```

14.5. Demo

This topic provides a demo on how to use Spark.

The demo is as follows:

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 History

14.6. Common cases

14.6.1. WordCount example

Spark runs simple WordCount.

Note

You must download the GitHub project and compile the project before running the corresponding 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 created. These JAR packages will be used to run the demos in this and the subsequent topics.

The following is the code for this 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(println)
    } 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

Spark can access OSS data.

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/write example

Read/write a MaxCompute table and convert it to Spark RDD.



Notice The project or table specified in the demo must exist or be changed to the specific project or 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,
            "dfctest_single_parted",
            Array("pt=20160101"),
            (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("p
t")))
        )
        // read from multi partition column table via rdd api
        val rdd_2 = odpsOps.readTable(
            projectName,
            "dfctest_parted",
            Array("pt=20160101,hour=12"),
            (r: Record, schema: TableSchema) => (r.getString(0), r.getString(1), r.getString("p
t"), 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("p
t1"), r.getString("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, sch

```

```
ema: TableSchema) => {
    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.jar
```

You can use either of the following methods to adjust the MaxCompute table read concurrency:

- Modify the `spark.hadoop.odps.input.split.size` parameter. A larger value results in a smaller number of 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

Use `sqlContext` to read/write a MaxCompute table.

Notice

- The project or table specified in the demo must exist or be changed to the specific project or 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. MaxCompute self-developed Console mode example

For safety reasons, machines in MaxCompute can not be directly connected. Therefore, the yarn-clientmode 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

Use PySpark to read/write MaxCompute tables.

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_test1"
    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

We recommend that you use OSS for read/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) + " fea
ture:" + 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

PySpark can only run on a host that can be directly connected to a computing cluster.

Install Python 2.7 and set the following path:

```
export PATH=/path/to/python/bin/:$PATH
```

Start command:

```
bin/pyspark --master yarn-client
```

Interactive execution:

```
df=spark.sql("select * from spark_user_data")
df.show()
```

14.6.9. Spark-shell interactive execution example (read tables)

Spark-shell can only run on a host that can be directly connected to a computing cluster.

Start command:

```
bin/spark-shell --master yarn
```

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 (MLlib and OSS read/write)

Spark-shell can only run 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=***
```

Start command:

```
bin/spark-shell --master yarn --jars cupid/hadoop-aliyun-package-3.0.0-alpha2-odps-jar-with-dependencies.jar
```

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. SparkR interactive execution example

SparkR can only be run on a host which can directly connect to a computing cluster. In addition, you must install R in the `/home/admin/R` directory and set the path:

```
export PATH=/home/admin/R/bin/:$PATH
```

Start command:

```
bin/sparkR --master yarn --archives ./R/R.zip
```

Interactive execution:

```
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. GraphX-PageRank example

Spark on MaxCompute supports native 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,hermesig,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 the 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. Spark Streaming - NetworkWordCount example

Spark on MaxCompute supports native Spark Streaming. To use NetworkWordCount, you need to install Netcat on your local host, and then run the following command:

```
$ nc -lk 9999
```

The input on the console then 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_S
ER)
    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 the 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

The GitHub project mentioned earlier can be used as your quick start template. For custom development, use the following pom.xml file.

Use Spark community edition 2.3.0 and ensure that 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 plugin has been released to the Maven warehouse. Add the following dependencies:

```
<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>
```

The file list in the Maven warehouse is as follows:

- The core code of the Cupid platform encapsulates the Cupid task submission API and the parent-child process read/write table API.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-core_2.11</artifactId>
  <version>1.0.0</version>
</dependency>
```

- The datasource encapsulates the spark-related MaxCompute table read/write API.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-datasource_2.11</artifactId>
  <version>1.0.0</version>
```

- The SDK encapsulates the 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 modes

Spark on MaxCompute supports three running modes: Local, Cluster, and DataWorks.

Local mode

The Local mode is used to facilitate code debugging for applications. In Local mode, you can use **Spark on MaxCompute** the same way as native Spark in the community. Additionally, you can use Tunnel to read and write data from and to MaxCompute tables.

In this mode, you can use either an IDE or the command line to run Spark on MaxCompute. If you use this mode, you must add the `spark.master=local[N]` configuration. N indicates the CPU resources required to implement this mode.

To use Tunnel to read and write data from and to tables in Local mode, you must add the Tunnel configuration item to `Spark-defaults.conf`. Enter the endpoint based on the region and network environment where the MaxCompute project is located.

The following code provides an example on how to use the command line to run Spark on MaxCompute in this 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 need to specify the Main method as the entry point of a custom application. The Spark job ends when Main succeeds or fails. This mode is suited for offline jobs. You can use Spark on MaxCompute in this mode together with DataWorks to schedule jobs.

The following code provides an example on how to use the command line to run Spark on MaxCompute in this 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** (in Cluster mode) in DataWorks to integrate and schedule the other types of nodes.

The following procedure is for reference only.

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. The resource file is the JAR file uploaded and published in the business flow.

You can specify configuration items, such as the number of executors and memory size, for the job to be submitted.

You also need to set `spark.hadoop.odps.cupid.webproxy.endpoint` to the endpoint of the region where the project is located, for example, `http://service.cn.maxcompute.aliyun-inc.com/api`.

4. You can manually run the Spark node to view the task operation log and obtain the URLs of LogView and JobView from the log for further analysis and diagnosis.

5. After you have defined the Spark job, you can orchestrate and schedule services of different types in the business flow.

14.8.2. Streaming tasks

MaxCompute also supports Spark Streaming. To support long-running tasks, add the following special configurations to `spark-defaults.conf`.

Compared with offline jobs, streaming jobs have additional configurations, which take effect immediately after they are completed in `spark-defaults.conf`.

```
spark.hadoop.odps.cupid.engine.running.type=longtime
-- Set the type to longtime so that the job will not be reclaimed.
spark.hadoop.odps.cupid.job.capability.duration.hours=25920
-- Set the duration.
spark.yarn.maxAppAttempts=10
-- Set the maximum number of retries for a failover.
spark.streaming.receiver.writeAheadLog.enable=true
-- Determine whether to enable the writeAheadLog mode. This mode prevents data loss but lowers the efficiency.
```

14.8.3. Job diagnosis

After you submit a job, you need to check the job log to determine whether the job is submitted and executed properly. MaxCompute provides LogView and Spark Web UI for you to diagnose jobs.

Submit a job in Spark-submit mode. Logs are also generated when you use DataWorks to execute Spark tasks. Example:

```
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.

```

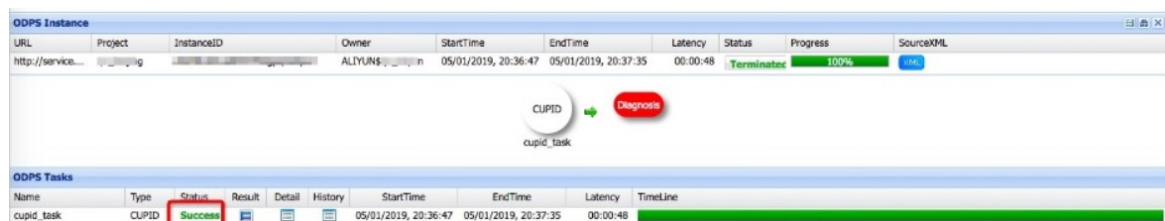
19/01/05 20:36:47 INFO YarnClientImplUtil: logview url: http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/api&p=qn_beijing&i=xxx&token=xxx
Success criterion: <If the following output is displayed, the operation succeeded. Other logs may be included in the output.>
19/01/05 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

```

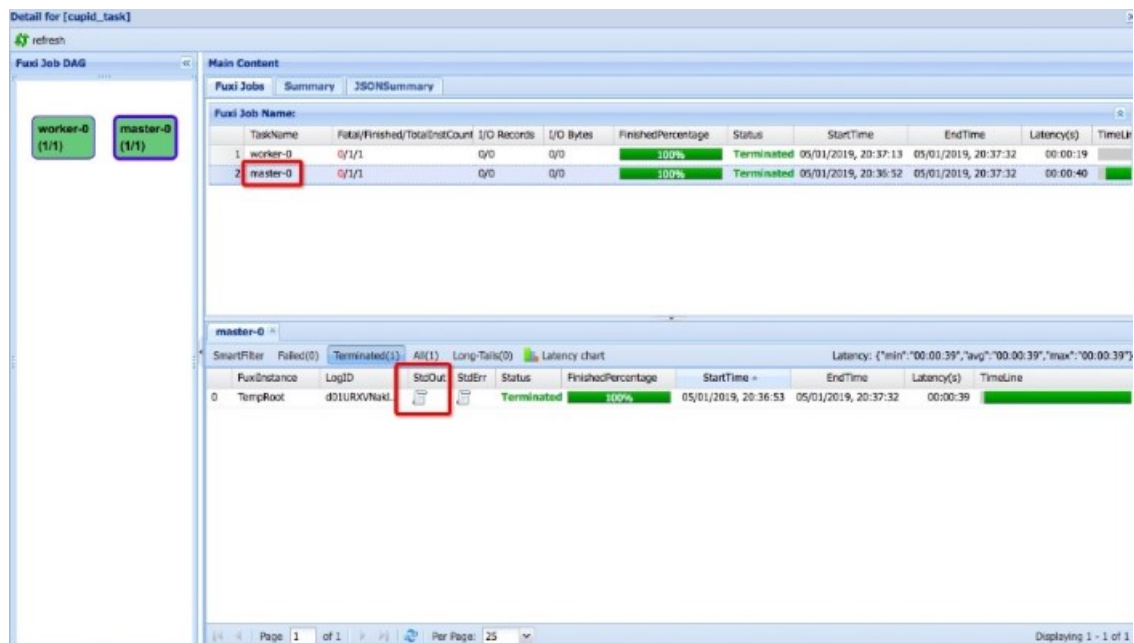
Use LogView to diagnose a job

Example:

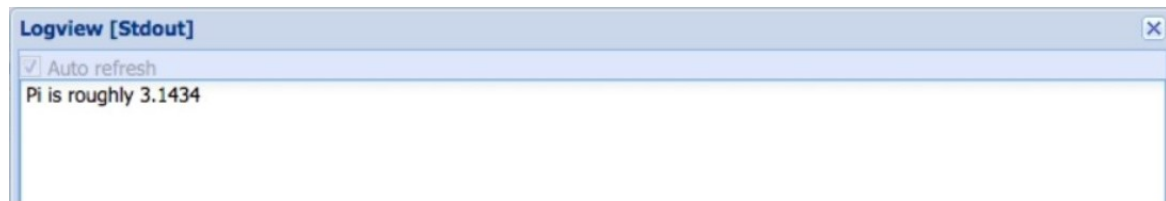
1. View basic execution information about the task of the CUPID type in a browser based on the LogView URL.



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 corresponding 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.

Example:

1. Access the URL in a browser to track the running status of your Spark job.

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

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 stderr stdout
1	worker50dcb649-cec8-4151-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	

2. Click **stdout** in the Logs column corresponding to driver to view the output of the Spark job.



14.9. APIs supported by Spark

14.9.1. Spark Shell

Run the following commands to start the application.

```
$cd $SPARK_HOME
-- Access the spark directory.
<b>Start command</b>: <codeblock>bin/spark-shell --master yarn</codeblock>
-- Select a 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

Run the following commands to start the application:

```
$mkdir -p /home/admin/R && unzip . /R/R.zip -d /home/admin/R/  
-- Create directory R and decompress R.zip in the directory.  
$export PATH=/home/admin/R/bin/:$PATH  
-- Set environment variables.  
$bin/sparkR --master yarn --archives . /R/R.zip  
-- Select a 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

Run the following commands to start the application:

```
$cd $SPARK_HOME  
-- Access the spark directory.  
$bin/spark-sql --master yarn  
-- Select a running mode and start the application.
```

Example:

```
show tables;  
select * from spark_user_data limit 3;;  
quit;
```

14.9.4. Spark JDBC

Run the following commands to start an application:

```
$sbin/stop-thriftserver.sh  
-- Stop a thread.  
$sbin/start-thriftserver.sh  
-- Restart a thread.  
$bin/beeline  
-- Start an application.
```

Example:

```
! connect jdbc:hive2://localhost:10000/odps_smoke_test
show tables;
select * from mr_input limit 3;
! quit
```

14.10. Spark dynamic resource allocation

Background

Spark provides a large number of configuration items to implement a wide array of semantics. You can use the default values for most configuration items, but there are some items require manual configurations. Of these items, **spark.executor.instances** is the most complicated.

A value that is too small will cause operations to run slowly or fail, while a value that is too large will waste resources.

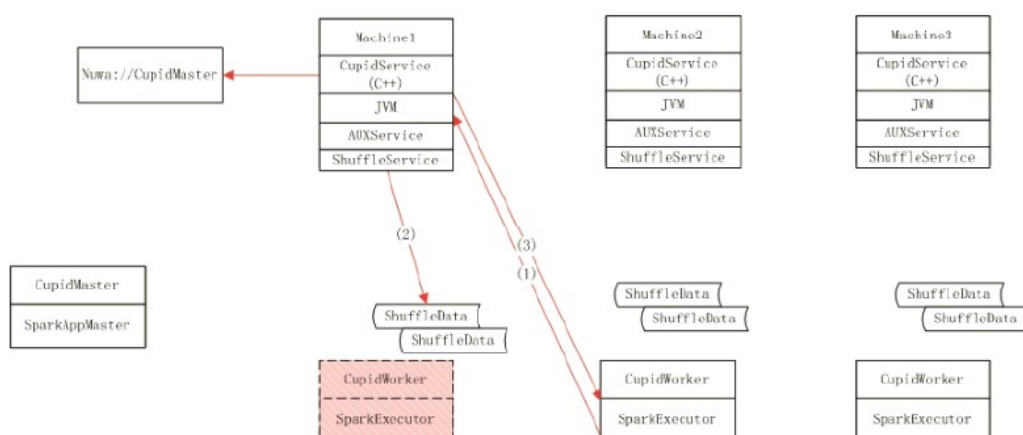
Even the optimal value based on full understanding of the data and logic is not reliable. For complex jobs, the number of executors required at different stages is different and different resources are required during the job execution process. Fixed configurations of resources will cause a waste. When long tail latency occurs, idle resources will be occupied by other executors even for simple jobs.

Solution

The best solution is to allocate resources as needed. Dynamic Resource Allocation (DRA) is such a solution. The CupidService developed by the Cupid team supports native DRA. Its implementation process is as shown in the following figure.

 **Note** For more information about the community native DRA and related configurations, see [Dynamic Resource Allocation in Job Scheduling](#) and [Dynamic Allocation in Spark Configuration](#).

Implementation process



Enable DRA

You must add the following settings to enable DRA. You do not need to modify the code.

```
spark.hadoop.odps.cupid.shuffleservice.enable=true // Required, the flag at the Cupid end.
spark.hadoop.odps.cupid.disk.driver.enable=true // Required, the dependent item.
spark.dynamicAllocation.enabled=true // Required, the flag at the Spark end.
spark.shuffle.service.enabled=true // Required, the dependent item.
spark.shuffle.service.port=7338 // Required, the shuffle service port.
spark.authenticate=true // Required, authentication.
spark.dynamicAllocation.maxExecutors=128 // Optional, the maximum number of executors.
spark.dynamicAllocation.minExecutors=1 // Optional, the minimum number of executors.
spark.dynamicAllocation.initialExecutors=1 // Optional, the initial number of executors.
spark.dynamicAllocation.executorIdleTimeout=60s // Optional, the waiting period before idle
executors are released.
```

When DRA is enabled, `spark.executor.instances` is optional and equivalent to

```
spark.dynamicAllocation.initialExecutors .
```

14.11. FAQ

This topic describes the frequently asked questions about Spark on MaxCompute.

How do I migrate open-source Spark code to Spark on MaxCompute?

The migration method depends on whether the job needs to access MaxCompute tables or OSS:

- If the job does not need to access MaxCompute tables or OSS:
Run the JAR package directly. Note that you must set the dependency on Spark or Hadoop to provided.
- If the job needs to access MaxCompute tables:
Configure the related dependencies and re-package the application.
- If the job needs to access OSS:
Configure the related dependencies and re-package the application.

How do I use Spark to access services in a VPC?

Currently, Spark does not allow access to services in a VPC. If you need to access a service in a VPC, submit a ticket to contact the MaxCompute team.

How do I troubleshoot the error indicating the ID and key provided by spark-defaults.conf are incorrect?

Error:

```
Stack:
com.aliyun.odps.OdpsException: ODPS-0410042:
Invalid signature value - User signature dose not match
```

Check whether the ID and key provided by spark-defaults.conf are consistent with the **AccessKey ID** and **AccessKey secret** obtained from the Apsara Stack console.

How do I troubleshoot the error indicating that I do not have permissions?

Error:

```
Stack:
com.aliyun.odps.OdpsException: ODPS-0420095:
Access Denied - Authorization Failed [4019], You have NO privilege 'odps:CreateResource' on
{acs:odps*:projects/*}
```

Contact the project owner to grant you the permissions to read and create resources.

How do I troubleshoot the error indicating that the project does not support Spark tasks?

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 is located. Check whether the configuration of `Spark-defaults.conf` is consistent with the requirements in the service documentation. If the Spark on MaxCompute service is provided in the region and the configuration of `Spark-defaults.conf` is correct, submit a ticket.

How do I handle the "No space left on device" error reported while a task is running?

Error:

```
No space left on device
```

Spark 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 online storage space is specified by the `spark.hadoop.odps.cupid.disk.driver.device_size` parameter. The default space is 20 GB and the maximum space is 100 GB. If this error persists after you adjust the capacity to 100 GB, further analysis is needed.

The most common cause is data skew. During the shuffle and cache processes, data is centrally distributed in some blocks. In this case, you can decrease the number of concurrent tasks in each executor (`spark.executor.cores`) or increase the number of executors (`spark.executor.instances`).

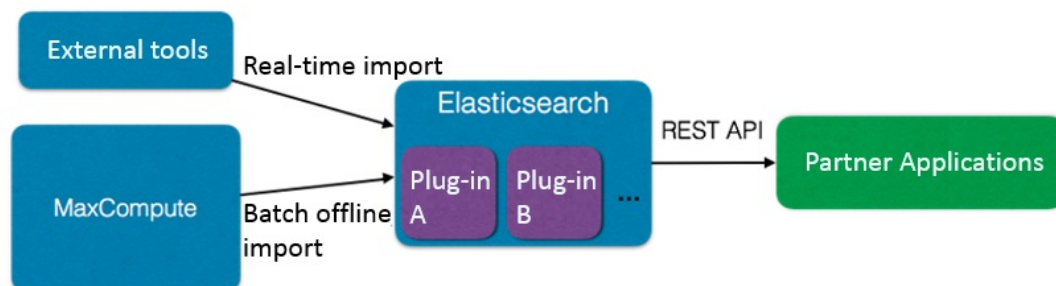
15.Elasticsearch on Maxcompute

15.1. Overview

Elasticsearch on MaxCompute is an enterprise-class full-text retrieval system developed by Alibaba Cloud for retrieving large volumes of data. It provides near-real-time (NRT) search performance for government agencies and enterprises. Elasticsearch on MaxCompute provides elastic full-text retrieval and supports native Elasticsearch APIs. Based on the APIs, it supports data import from heterogeneous data sources as well as cluster and service OAM. Based on the centralized scheduling and management capabilities of MaxCompute, Elasticsearch provides more efficient core services for retrieving large volumes of data. Elasticsearch on MaxCompute can also work with plugins available from the Elasticsearch open source community to provide a range of retrieval features.

You can import data to Elasticsearch on MaxCompute using external tools in real time or use MaxCompute to import offline data in batches. After indexing imported data, Elasticsearch on MaxCompute provides the retrieval service through the RESTful API. The following figure shows the usage of Elasticsearch on MaxCompute.

Elasticsearch on MaxCompute usage



15.2. Workflow

15.2.1. Overview

Elasticsearch on MaxCompute is 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 needed, including the number of nodes, disk space, memory size, and custom settings. The resources consumed by the Elasticsearch service are counted towards your MaxCompute instance quota.

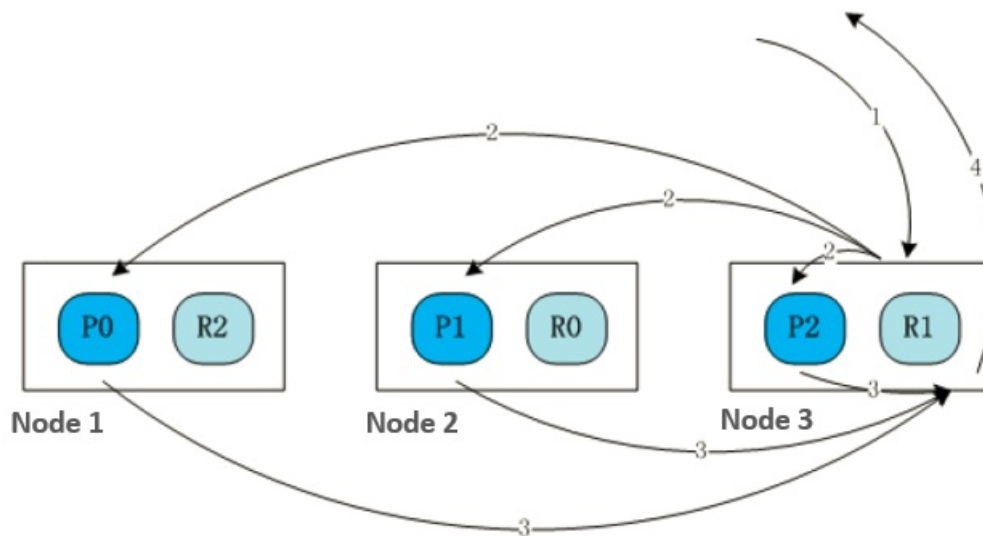
For the process of starting Elasticsearch, see the workflow chart in [Typical practice of Elasticsearch](#).

The following topics describe the workflows of Elasticsearch on MaxCompute features.

15.2.2. Distributed retrieval workflow

The following figure shows the distributed retrieval workflow:

Distributed retrieval workflow



Each cluster consists of three nodes, as shown in the preceding figure. The index has three shards: P0, P1, and P2, which are distributed across three nodes. The shards work in 1:1 backup mode, so there are three replicas: R0, R1, and R2. The retrieval process is as follows:

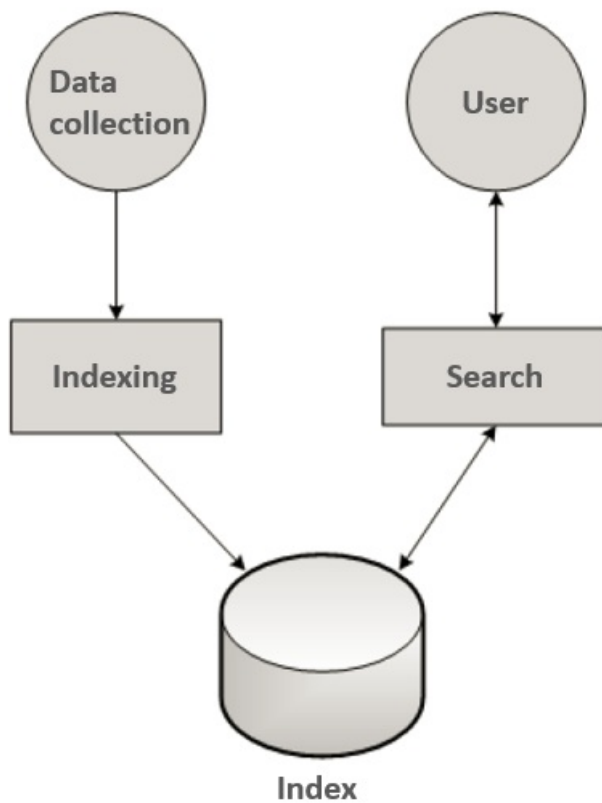
1. A user sends a retrieval request to node 3.
2. After receiving the request, node 3 sends a retrieval request (2) to P0, P1, and P2 based on the recorded index shard information.
3. The nodes where P0, P1, and P2 are located search for the requested information in the specified shards. Each node sends a retrieval result message (3) to node 3.
4. Node 3 combines the retrieval results from the other nodes, and returns a result to the user in an acknowledgment message (4).

Because multiple nodes perform data retrieval at the same time, the retrieval speed is improved. The performance of distributed retrieval increases with the number of nodes.

15.2.3. Full-text retrieval process

The following figure shows the full-text retrieval process.

Full-text retrieval process



The process is as follows:

1. The data collection module collects structured and unstructured data, converts the data into the field + value format, and submits the data to the index module.
2. The index module receives the data in the field + value format, performs segmentation and creates inverted indexes based on the predefined indexing method, and saves the indexes. The field type, indexing method, and segmentation rule are configured on the retrieval management page.
3. The search module receives and processes user requests. It parses the requests to obtain indexes, fields, and query statements. Then, the search module finds matching records from the inverted indexes.
4. The search module returns data that satisfies the user requirements, such as the sorting rule and quantity.

15.2.4. Authentication process

The authentication process is as follows:

1. **You try to log on to the Elasticsearch on MaxCompute** retrieval management or O&M platform. You are redirected to the authentication module. If you pass the authentication, you can access the platform. Otherwise, you are denied access to the platform.
2. The administrator can use the MaxCompute client to add Elasticsearch users and configure permissions for the users.
3. When you try to access the index library through an API, the system implements authentication. You can search for or operate data in the index library only after passing the authentication.

15.3. Quick start

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, 8 GiB of memory and 20 GiB of disk space are allocated to each node.

 **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 shows how to quickly start a small Elasticsearch cluster based on the default configurations. The name of the Elasticsearch cluster is `es_first_cluster`.

1. Download the [MaxCompute client](#) that supports Elasticsearch. Configure the AccessKey pair, project, and endpoint.
2. Start the client (`odpscmd`) and run the following command:

```
server create es_first_cluster type elasticsearch_mdu;
```

Wait for several minutes. If OK is returned, the Elasticsearch cluster is active. You must create an Elasticsearch user to access the cluster.

3. In `odpscmd`, create an Elasticsearch user with the permission of `all_access`.

```
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. Assume that the `prj1` MaxCompute project is used to access the Elasticsearch cluster. Run the following command to return information about the Elasticsearch cluster:

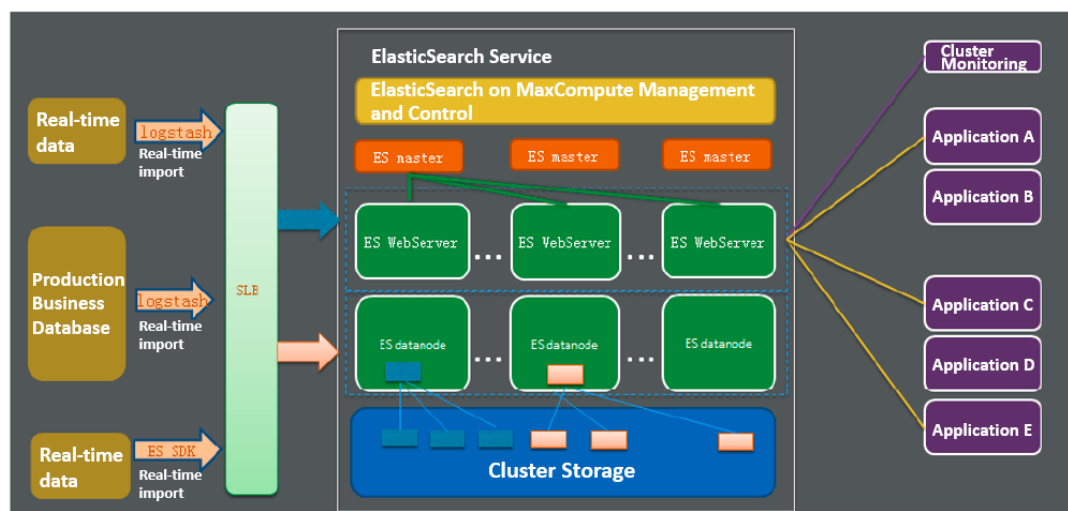
```
curl -u admin:123456 http://search.aliyun.com:9200/prj1.es_first_cluster
```

 **Note** 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 recovered.

15.4. Support for Elasticsearch applications

15.4.1. ElasticSearch typical practice

Typical practice



Elasticsearch on MaxCompute allows you to start a set of Elasticsearch services on MaxCompute clusters by submitting a job. The project does not modify the native Elasticsearch code. Elasticsearch on MaxCompute works in the same way as the native Elasticsearch cluster.

15.4.2. Elasticsearch on MaxCompute support for VPC

Alibaba Cloud Elasticsearch on MaxCompute is an enterprise-class full-text retrieval system for retrieving massive amounts of data. To comply with data isolation and security requirements, Elasticsearch on MaxCompute provides support for Virtual Private Cloud (VPC) networks so that you can apply access policies at VPC level. (Elasticsearch VPC limits).

Elasticsearch on MaxCompute supports VPC networks in the following model:

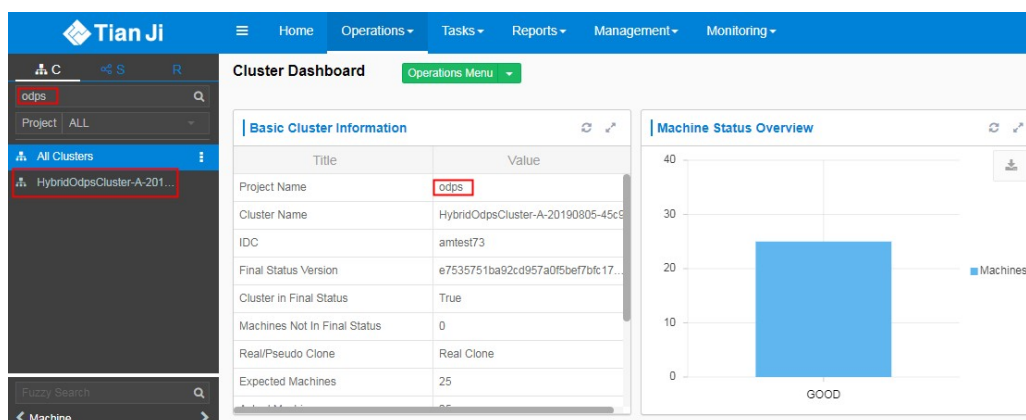
- Classic networks, VPC, and the Internet are isolated from each other. Users can access only the endpoints and virtual IP addresses (VIPs) on their networks.
- Projects without a whitelist of VPC IDs and IP addresses are accessible for users from valid domains over the three types of networks. A domain is valid only if its access request is acknowledged.
- When an Elasticsearch service instance is started in a MaxCompute project, they share the same VPCLIST, which is a whitelist of VPCs.
- Starting an Elasticsearch service instance occupies all resources by default. You must scale up the MaxCompute instance or scale down the Elasticsearch service instance if you start more Elasticsearch service instances.

Here is a specific use example. Starting one Elasticsearch service instance for each project is taken as the default practice when a MaxCompute VPC is deployed. You can start your Elasticsearch instances for your projects, apply for domain names and VIPs, and perform VPC health check in the Elasticsearch frontend.

15.5. Special notes

15.5.1. Find the Elasticsearch service domain name

1. Log on to the Apsara Infrastructure Management Framework console. Choose Operations > Cluster Operations from the top navigation bar. In the left-side navigation pane, click the C tab and search for ODPS.



2. Navigate to the MaxCompute cluster resource usage page.
3. Find the Elasticsearch service domain name.

service	serverrole	app	n...	ty...	s...	...	result
odps-service-computer	odps-service-computer.Com...	com...	odps	ots	done	{ "e...	{ "instance_name": "odps", "db_name": "TIANJIA-A2B4", "db_user": "1067309636274922", "db_f...
odps-service-console	odps-service-console.Cupid...	cupi...	odps...	dns	done	{ "d...	{ "ip": "10.36.103.41", "domain": "jobview.cn-hangzhou-env6-d01.odps.aliyun-inc.com", "dns...
odps-service-console	odps-service-console.LogVie...	log_v...	odps...	dns	done	{ "d...	{ "ip": "10.42.36.0.208", "domain": "logview.cn-hangzhou-env6-d01.odps.aliyun-inc.com", "dns": "lo...
odps-service-console	odps-service-console.WebC...	web...	odps...	dns	done	{ "d...	{ "ip": "10.36.103.155", "domain": "webconsole.cn-hangzhou-env6-d01.odps.aliyun-inc.com", "dns": "we...
odps-service-console	odps-service-console.WebC...	web...	odps...	dns	done	{ "d...	{ "ip": "10.42.36.0.210", "domain": "webconsole.cn-hangzhou-env6-d01.odps.aliyun-inc.com", "dns": "we...
odps-service-es	odps-service-es.ElasticSearc...	elasti...	odps...	dns	done	{ "d...	{ "ip": "10.36.103.65", "domain": "elasticsearch.cn-hangzhou-env6-d01.odps.aliyun-inc.com", "dns": "el...
odps-service-frontend	odps-service-frontend.Fronte...	front...	odps...	dns	done	{ "d...	{ "ip": "10.36.102.181", "domain": "service.cn-hangzhou-env6-d01.odps.aliyun-inc.com", "dns": "se...
odps-service-frontend	odps-service-frontend.Tunnel...	tunn...	odps...	dns	done	{ "d...	{ "ip": "10.36.102.177", "domain": "dt.cn-hangzhou-env6-d01.odps.aliyun-inc.com", "dns": "dt.cn...

15.5.2. Import table data from MaxCompute to Elasticsearch

Before you can use **Elasticsearch on MaxCompute** to search for MaxCompute table data, you must import table data from MaxCompute to Elasticsearch clusters. MaxCompute MapReduce is designed to carry out the task of exporting data from MaxCompute to Elasticsearch. You can import data to Elasticsearch through simple configurations.

By utilizing the distributed dispatching capability of MaxCompute, you can easily control the concurrency. Besides, you can add the MapReduce job to the scheduled tasks on D2.

The data import process is as follows:

1. Download the JAR package of the MapReduce job.
2. Run the following command to add the JAR package to the MaxCompute resource files in the MaxCompute console:

```
add jar /PATH/TO/elasticsearch_output-1.0.0.jar
```

3. Create configuration file `es_mr.conf` in the following format for the MapReduce job:

```
<configuration>
  <property>
    <name>key1</name>
    <value>value1</value>
  </property>
  <property>
    <name>key2</name>
    <value>value2</value>
  </property>
</configuration>
```

4. Submit the MapReduce job in the MaxCompute console. Example:

```
jar -conf es_mr.conf -classpath /PATH/TO/elasticsearch_output-1.0.0.jar
    -resources elasticsearch_output-1.0.0.jar -Dworker_num=5
com.aliyun.odps.export.elasticsearch.mr.EsOutputJob <TABLE_NAME> [PARTITION_SPEC];
-- Five workers run concurrently to export data from <TABLE_NAME> [PARTITION_SPEC] to t
he Elasticsearch cluster.
```

The following table describes the parameters in the es_mr.conf file.

Parameters

Parameter	Example	Required	Default value	Description
es.resource	my_index/my_type	Yes	-	Index and type in the target Elasticsearch for imported data.
es.nodes	-	Yes	-	Elasticsearch endpoint.
es.nodes.client.only	true	No	false	Send data to client-only nodes only.
es.col.field.mapping	odps_col1:es_field1,odps_col3:es_field2	Yes	-	Mapping between the MaxCompute columns to be imported and the Elasticsearch fields.
es.batch.size.bytes	1 MB	No	1 MB	Batch data transfer size.
es.batch.size.entries	1000	No	1000	Number of data entries transferred each time.
es.net.http.auth.user	admin	No	-	Elasticsearch username.
es.net.http.auth.pass	123456	No	-	Elasticsearch password.
es.mapping.routing	field_routing	No	-	Routing field name, in the <CONSTANT> format for a constant.
es.mapping.id	field_id	No	-	id field of a document.

16. Flink on MaxCompute

In the current MaxCompute version, **Flink on MaxCompute** is only for trial use. The following 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 configuration items can be obtained from the *odps_config.ini* file under the */home/admin/odps/odps_tools/clt/conf/* directory.
- *odps.cupid.proxy.domain.name* specifies the domain name of *odps_jobview_server_dns*. The value is the domain name without *sparkui*.

2. Start the program.

Save *flink-java-project-0.1.jar* 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.

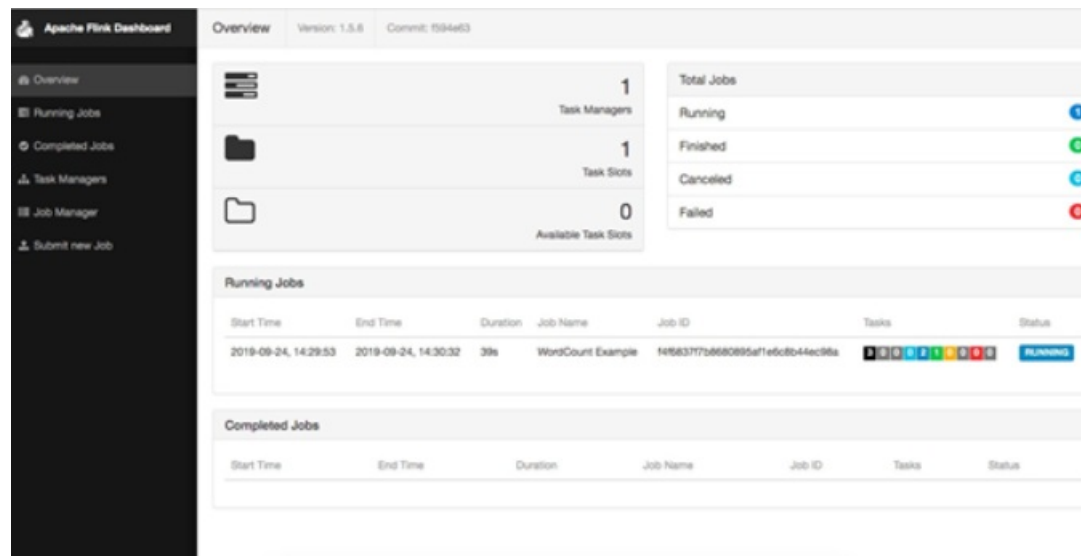
Logview

The screenshot shows the MaxCompute Logview interface. At the top, there's a 'Welcome, Guest' message. Below it, the 'OOPS Instance' table is displayed with columns: URL, Project, InstanceID, Owner, StartTime, EndTime, Latency, Status, Priority, SourceURL, and Tool. A single instance is listed with status 'Running'. Below the instance table, there are two buttons: 'CUPID' and 'Diagnose'. Underneath these buttons is the 'OOPS Tasks' table with columns: Name, Type, Status, Result, Detail, History, StartTime, EndTime, Latency, and TimeLine. A task named 'cupid_task' is listed with status 'Running'.

OOPS Instance										
URL	Project	InstanceID	Owner	StartTime	EndTime	Latency	Status	Priority	SourceURL	Tool
http://service...	odps_smo...	2019054062...	ALYUApas...	24/05/2018, 14:29:47	-	00-01:15	Running	1		No Link (not internal project)

OOPS Tasks									
Name	Type	Status	Result	Detail	History	StartTime	EndTime	Latency	TimeLine
cupid_task	CUPID	Running				24/05/2018, 14:29:47	-	00-01:15	

Flink



Note MaxCompute is compatible with Flink in terms of the following aspects:

- Group windows of Flink: The TUMBLE, HOP, and SESSION built-in functions are supported.
- Flink UDFs, UDAFs, and UDTFs: MaxCompute supports UDFs that run in Flink. You can use the `odps.sql.enable.flink.udf` flag to enable this feature.
- SQL syntax in Flink: The `||` operator is supported to connect strings. `extract(dateunit from datetime)` and `substring(str from startIndex to endIndex)` are supported.
- Flink HistoryServer: You can use Flink HistoryServer to view information about completed tasks.

17. Non-structured data access and processing (integrated computing scenarios)

17.1. Overview

MaxCompute SQL cannot directly process external data, such as non-structured data from Object Storage Service (OSS). This type of data must be imported to MaxCompute tables by using the relevant tools, which 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 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 like 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, ApsaraDB for RDS, Alibaba Cloud HDFS, and TDDL
- External data sources: open source HDFS, MongoDB, and 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. Preface

As the core computing component of the Alibaba Cloud big data platform, MaxCompute provides powerful computing capabilities. It can schedule large amounts of nodes for parallel computing, and effectively manage failover and retry mechanisms in distributed computing. MaxCompute SQL implements different data processing logics through simple semantics. It is widely used within and outside Alibaba Group. It allows interoperability among different data sources and is significant for the integrated data ecology of Alibaba Cloud.

The following examples show how to access and process OSS unstructured data in MaxCompute.

17.2.1.2. Use the built-in extractor to read OSS data

17.2.1.2.1. Overview

You can use the MaxCompute built-in extractor to easily read OSS data in the specified format. You only need to create an external table as the source table for data query. 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 file path is `/demo/SampleData/CSV/AmbulanceData/vehicle.csv`. The following topics provide operation examples.

17.2.1.2.2. Create an external table

Execute the following statement 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'
LOCATION 'oss://<your-id*>:<your-secret-key*>@oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/SampleData/CSV/AmbulanceData/';
```

Note

- `com.aliyun.odps.CsvStorageHandler` is a built-in storage handler that is used to process CSV files. It defines how to read and write CSV files. You only need to specify this parameter. The logic is implemented by the system. To read and write TSV files, use `com.aliyun.odps.TsvStorageHandler`.
- `LOCATION` specifies an OSS directory. The system reads all files in the directory by default.
- An external table contains only related OSS directories. If you delete this table, the data in the directory specified by `LOCATION` is not deleted.

17.2.1.2.3. Query an external table

After an external table is created, you can use it the same way that you use a standard table.

The following content is included in the `/demo/SampleData/CSV/AmbulanceData/vehicle.csv` file:

```
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 only be managed by using MaxCompute SQL, not MaxCompute MapReduce.
- To obtain data over HTTPS at the underlying layer, add the `set odps.sql.unstructured.data.oss.use.https=true;` flag 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 built-in CsvStorageHandler, TsvStorageHandler, and TextStorageHandler.

17.2.1.2.4. MSCK REPAIR TABLE

The **MSCK REPAIR TABLE** statement can be used to add partitions to external tables. Syntax:

```
MSCK [REPAIR] TABLE external_table_name [ADD PARTITIONS];
```

To add partitions, perform the following steps:

1. Import data to Object Storage Service (OSS). The storage path must be in the following format: `oss://xxx/table-location/ptname1=ptvalue1/ptname2=ptvalue2/xxx`.
2. Create an external table and specify the structures of the ptname1 and ptname2 partitions.
3. Execute the `msck repair table external_table_name [add partitions]` statement. MaxCompute SQL automatically parses the OSS directory structure, identifies partitions, and adds partitions to the external table.

Sample code:

```
CREATE EXTERNAL TABLE orc_pt_v0
(
  name string
)
partitioned by (pt bigint)
STORED AS textfile
location 'oss://xxx/odps-ext-reg-perf/orc-pt-v0';msck repair table orc_pt_v0 add partitions
;
-- In this case, the MSCK 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);
```

17.2.1.3. Custom extractors

17.2.1.3.1. Overview

If OSS data is in a complicated format that cannot be processed by the built-in extractor, you must customize an extractor to read data from OSS files. For example, a text file is stored in OSS. The file is not in the CSV format and the columns of records are separated by vertical bars (|). The file path is `/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv`. The following topics provide operation examples.

17.2.1.3.2. Define StorageHandler

You can customize the data parsing logic. StorageHandler is the unified entrance of your custom logic. You can specify the types of custom extractors and outputers. StorageHandler provides only a simple definition. For example, you can implement 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 Outputer> getOutputerClass() {
    return TextOutputer.class;
  }
}
```

 **Note** Note that TextStorageHandler that is built in MaxCompute can process the data format in the preceding example (text separated by vertical bars (|)). This example is provided only to show you how to use the SDK to customize a StorageHandler (especially extractor) for processing uncommonly structured data.

17.2.1.3.3. Define an extractor

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 next() is calle
        d. This InputStream can read all the content of 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 initied 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 using the delimiter. For the impleme
        ntation process, see Complete TextExtractor implementation.
        -- 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

You can compile and package Java code, and run the following command to upload it to MaxCompute. The procedure is the same as that for a normal Java UDF.

```
add jar odps-udf-example.jar;
```

17.2.1.3.5. Create an external table

After you upload a JAR package, you need to run the following command to create an external table. This command is similar to the one that you run before using a built-in extractor. The difference is that this command uses a custom `StorageHandler`.

```
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 StorageHandler.
WITH SERDEPROPERTIES('delimiter'='|')
-- SERDEPROPERTIES can be used to specify parameters. These parameters are transferred to an extractor through 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 where the class definition is located.
```

17.2.1.3.6. Query an external table

The content of `/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv` is as follows:

```
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
```

Run the following command 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;
```

Command output:

```
+-----+-----+-----+
| 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. Advanced usage

17.2.1.4.1. Use a custom extractor to read external unstructured data

The preceding topic describes 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 UDF extractors to process non-text files in OSS.

The following example shows how to process audio files (.wav files) in OSS.

1. Customize the SpeechSentenceSnrExtractor main logic. Use the SETUP API to read parameters, initialize the parameters, and import the audio processing model (by using the resource function).

```
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
```

```

@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 `Extractor()` API 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 result to a record in `[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. The command output is as follows:

```
-----
| 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** By using the UDF extractor, you can run 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

In the preceding topic, LOCATION is used to specify an OSS directory, which is associated with an external table. MaxCompute reads all data in the directory, including all files in the subdirectories. When there is a large volume of data in the directory, a full-text scan will cause extra I/O operations and processing time. There are two solutions:

- Reducing the data volume: You need to properly plan data storage addresses. Create multiple external tables for data from different parts, with the LOCATION of each external table pointing to a subset of data.

- Partitioning data: The external table, like an internal table, supports partitioning. You can create partitions to facilitate data management.

The following topics describe the partition feature of external tables.

17.2.1.5.2. Standard organization method and path format of partition data in OSS

Unlike the data in internal tables, data stored in external storage (such as OSS) cannot be managed in MaxCompute. If you need to use the partitioned table feature of MaxCompute, make sure that the data file paths 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. For the partitions to take effect, you must specify the OSS storage directory in the format shown in the preceding example. An example of a valid directory layout is as follows:

```
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 have uploaded offline data to OSS by using osscmd or other OSS tools, you can define the data path format. To ensure the partitioned external table feature operates normally, we recommend that the path where data is stored to is in the format specified in the previous example.

- Then, you can execute the ALTER TABLE ADD PARTITION statement to import the partition information to MaxCompute. An example of the DDL statement is as follows:

```
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')
...
```

- When the data is ready and the partition information has been imported to MaxCompute, you can execute SQL statements to operate the partitions in the external table in OSS.
 - Execute the following statement to count the number of unique IP addresses in the log dated June 1, 2017:

```
SELECT count(distinct(ip)) FROM log_table_external
WHERE year = '2017' AND month = '06' AND day = '01';
```

Note In the log_table_external directory that corresponds to an external table, only files in the *log_data/year=2017/month=06/day=01* subdirectory (logfile and To prevents unnecessary I/O operations, a full scan of the *log_data/* directory is not performed.

- Similarly, you can execute the following statement to analyze data for the second half of 2017:

```
SELECT count(distinct(ip)) FROM log_table_external
WHERE year = '2017' AND month > '06';
```

Note In this case, only the logs for the second half of 2017 stored in OSS are accessed.

17.2.1.5.3. Custom path of partition data in OSS

If you have historical data stored in OSS paths that are not in the partitionKey1=value1\partitionKey2=value2\... format, you can still access the data by using the MaxCompute partition feature. MaxCompute provides a way to import partitions through a custom path.

Example:

- The data path only contains partition values (without partition keys). An example of the path layout is as follows:

```
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 statement 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/lo
g_data_customized/2017/06/01/';
```

 **Note** The ADD PARTITION and LOCATION clauses are specified in the preceding example to bind the partitions to data paths. Even if the data storage path is not in the partitionKey1=value1\partitionKey2=value2\... format, you can still access the partition data in the subdirectory.

17.2.1.5.4. Access fully-customized non-partitioned data subsets

In certain situations, you might need to access a file subset in an OSS path, but files in this subset do not have any obvious regularity in terms of directory layout. The unstructured data processing framework of MaxCompute is able to handle this situation, but will not be discussed in this topic.

If you require advanced operations such as this, contact the MaxCompute technical team for support.

17.2.1.6. Output OSS data

17.2.1.6.1. Create an external table

To write data to OSS, you need to run 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 run MaxCompute SQL statements such as INSERT INTO and INSERT OVERWRITE to write data to OSS. In the following example, the built-in TsvStorageHandler is used.

```
DROP TABLE IF EXISTS tpch_lineitem_tsv_external;
CREATE EXTERNAL TABLE IF NOT EXISTS tpch_lineitem_tsv_external
(
  orderkey BIGINT,
  supplekey 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 will be used to read or write data from or to the external table.
- Data storage format: `StorageHandler` is used to define the data access mode. In this example, MaxCompute built-in `com.aliyun.odps.TsvStorageHandler` is used to read or write data from or to TSV files. You can also use the MaxCompute SDK to define `StorageHandlers`.

17.2.1.6.2. Write data to a TSV text file by using an INSERT statement on an external table

After you associate a file in OSS with an external table, you can run a standard SQL `INSERT OVERWRITE/INSERT INTO` statement on the external table to write data to the OSS file. The source data can be either data stored in a MaxCompute internal table or external data that is imported to MaxCompute through an external table.

Note

- MaxCompute internal table: You can run an `INSERT` statement on an external table to write data from a MaxCompute internal table to an external storage medium.
- External data imported to MaxCompute through an external table: You can import external data to MaxCompute through an external table, use the data for computations, and then export the results to an external address or storage medium. For example, import Table Store data to MaxCompute and then export the data to OSS.

The following example assumes that you have a MaxCompute internal table named `tpch_lineitem` and want to export some of the data to OSS in the TSV format. After you create an external table, run the following `INSERT OVERWRITE` statements to export 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;
```

The preceding example selects eight columns from the rows in `tpch_lineitem` table that satisfy the conditions `l_discount = 0.07` and `l_tax = 0.01` and writes it to `tpch_lineitem_tsv_external` in OSS in the TSV format. After this operation is complete, you can view the corresponding TSV data file in OSS.

Notice

Data exported from MaxCompute to OSS is stored in a special file structure.

- When you run `INSERT INTO/OVERWRITE` statements on an OSS address, all data is exported to the `.odps` folder at the specified `LOCATION`.
- The `.meta` file in the `.odps` folder is an extra macro data file written by MaxCompute to record valid data in the current folder. Typically, if the `INSERT` operation is successful, all the data in the current folder is valid. You are only required to parse the macro data if a job fails.
- If a job fails or is terminated, perform the `INSERT OVERWRITE` operation again until it is complete. This prevents parsing of the `.meta` file.
- If you need to parse the `.meta` file, contact Alibaba Cloud technical team for support.

The number of files that are generated during MaxCompute built-in TSV/CSV processing 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 need to force a TSV file to be generated, you can append `DISTRIBUTE BY l_discount` to the `INSERT OVERWRITE` operation. Then, a reduce stage with only one reducer is added at last so that only one TSV file is output.

17.2.1.6.3. Write data to an unstructured file by using an INSERT statement on an external table

MaxCompute also provides Outputter APIs for data output. You can use the APIs to write user data to a custom unstructured data file through `OutputStream`. Further details are not covered in this topic.

If you have this requirement, contact the MaxCompute technical team for support.

17.2.1.6.4. Migrate data between different storage media with MaxCompute

External tables act as an interface between MaxCompute and external storage media. External tables can be used to read or write data from or to various external storage media such as OSS and Table Store. Based on the external table feature, various data computing and storage links can be established. For example,

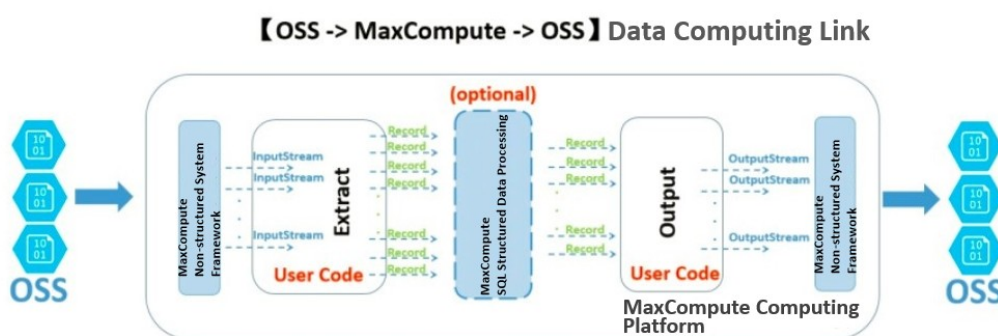
1. MaxCompute reads the OSS data associated with External Table A, and performs complicated computations. MaxCompute then outputs the results to the OSS address associated with External Table B.

2. MaxCompute reads the Table Store data associated with External Table A, and performs complicated computations. MaxCompute then outputs the results to the OSS address associated with External Table B.

Note The preceding examples and data sources are scenarios with MaxCompute tables. The only difference is that the SELECT statement originates from an external table instead of a MaxCompute table.

Example:

By using MaxCompute as a central computing platform, you can import data from an OSS instance, and export that data to a different OSS instance (in a different location, or a different OSS account), as shown in the following figure.



From a data flow and processing logic standpoint, the unstructured data processing framework can be considered as a coupled data ingress and egress at both ends of the MaxCompute platform.

1. The external data (from an OSS instance) is converted based on the unstructured framework, and provided to the UDF API in the form of a Java InputStream class. The UDF extract logic is only required 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 built into MaxCompute, and may generate new records.
3. The computed records are transferred to the UDF Output logic for further computation. Finally, the required information is extracted from the records, output through OutputStream, and written to the OSS instance.

Notice You can perform any combination of the preceding steps based on your needs.

17.2.1.7. STS mode authorization for OSS

When you create an external table, the Location-based OSS access account supports plaintext input of the AccessKey ID and AccessKey secret, but the account information may be exposed. To prevent the leakage of account information, MaxCompute provides a more secure way to access OSS.

MaxCompute combines RAM and Security Token Service (STS) of Alibaba Cloud to resolve security issues of accounts. You can grant permissions in two ways:

- 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 set the policy content as follows:

```
-- If the owners of MaxCompute and OSS use the same account:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}

-- If the owners of MaxCompute and OSS use different accounts:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ID of the Alibaba Cloud account used by the owner of MaxCompute@odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

- ii. Set the `AliyunODPSRolePolicy` permission, which is the necessary permission 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 need to specify the RAM information when you create an OSS external table later.

17.2.2. Table Store data source

17.2.2.1. Preface

As the core computing component of the Alibaba Cloud big data platform, MaxCompute meets most distributed computing requirements within and outside Alibaba Group. As the entry of distributed data processing, MaxCompute SQL provides powerful support for quick processing and storage of large volumes (exabytes) of offline data. With the continuous expansion of big data business, 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 in detail how to import data from Table Store to MaxCompute, implementing seamless interoperability between data sources.

Compared with traditional databases, NoSQL KV Store supports flexible schema, scalability and real-time response for applications such as online service. Alibaba Cloud Table Store is a large-scale NoSQL data storage service based on the Apsara system. It supports storage and real-time access of massive KV data. Table Store is widely used by all business units in Alibaba Group and the Alibaba Cloud ecosystem. In particular, Table Store features, such as row-level real-time update and override writing, are a supplement to the append-only operation of MaxCompute tables. As a storage-oriented service, Table Store does not provide sufficient computing capabilities to process large amounts of data concurrently. This makes it important to enable data interoperability between MaxCompute and Table Store.

The following examples show how to access and process Table Store data in MaxCompute.

17.2.2.2. MaxCompute reads and computes data in Table Store

17.2.2.2.1. Prerequisites and assumptions

This document assumes that you have basic knowledge of Table Store operations. If you are not familiar with Table Store or KV tables, we recommend that you first familiarize yourself with the basic concepts of Table Store (such as primary keys, partition keys, and attribute columns) before reading this topic.

17.2.2.2.2. Create an external table

External tables can act as interfaces between MaxCompute and Table Store: You can run a DDL statement (CREATE EXTERNAL TABLE) to import a table description in Table Store to the MaxCompute meta system. Then, you can process data in the Table Store table in the same way you process data in a MaxCompute table.

Example:

```

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 StorageHandler for processing Table Store data. It defines the interaction between MaxCompute and Table Store. The relevant logic is implemented by MaxCompute.
WITH SERDEPROPERTIES (
  -- SERDEPROPERTIES is an API that provides parameter options. Two options must be specified for TableStoreStorageHandler: tablestore.columns.mapping and tablestore.table.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 Table Store tables that are accessed by MaxCompute, including primary key and attribute columns. Column names starting with a colon (:) are primary key columns in Table Store tables. In this example, :o_orderkey and :o_orderdate are primary key columns. The other column names specified are attribute columns. Table Store supports up to four primary keys of the bigint or string type. The first primary key is the partition key. When you specify a mapping, you must provide all primary key columns of the specified Table Store table. You do not have to specify all the attribute columns, only those accessed by MaxCompute.
  'tablestore.table.name'='ots_tpch_orders'
  -- tablestore.table.name: This option is required. It describes the names of Table Store tables that are accessed by MaxCompute. If you specify an invalid (nonexistent) Table Store table name, an error is returned and MaxCompute does not create a Table Store 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 Table Store information, including the instance name and endpoint.

```

 **Note** The preceding example maps a Table Store table to a MaxCompute external table. The subsequent operations on the Table Store table can be performed through the external table.

17.2.2.2.3. Access Table Store data through an external table

After you follow the preceding example to create an external table, Table Store data is imported to MaxCompute. Then, you can access Table Store data by using MaxCompute SQL statements.

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** This example uses common MaxCompute SQL statements. Table Store access details are processed internally by MaxCompute.

If you need to use a copy of data for multiple computations, you can import the data from Table Store to a MaxCompute table (internal). This is more efficient than reading the data from Table Store every time.

Example:

```
CREATE TABLE internal_orders AS
SELECT odps_orderkey, odps_orderdate, odps_custkey, odps_totalprice
FROM ots_table_external
WHERE odps_orderkey > 5000 ;
```

 **Note** internal_orders is a common MaxCompute table that has all the features of a MaxCompute internal table. These features include efficient compressed column storage and complete meta. This table is stored in MaxCompute, so it can be accessed faster than an external table in Table Store. This feature is particularly suitable for hot data that is used for multiple computations.

17.2.2.3. Export data from MaxCompute to Tablestore

Data interaction between MaxCompute and Tablestore includes importing data from Tablestore to MaxCompute for batch processing and exporting 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 create external tables in Tablestore. Before you export data to a table in Tablestore, make sure that the table exists in Tablestore. Otherwise, an error is reported.

Assume that a user creates the ots_table_external table 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 internal_orders data record and write the processing result to Tablestore, execute the INSERT OVERWRITE TABLE statement. Example:

```
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 data storage 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.
- When you execute the INSERT OVERWRITE statement on the external table, MaxCompute writes 200 data records in each batch 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

AnalyticDB updates or processes data. If both the data processed by AnalyticDB and the data in MaxCompute are used for computation, the data from AnalyticDB must be synchronized with the data from MaxCompute. To accomplish this, you can create an external table to access the AnalyticDB data.

The following example shows how MaxCompute accesses and processes AnalyticDB data.

17.2.3.2. Write data to AnalyticDB

17.2.3.2.1. Create an external table

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}'
TBLPROPERTIES(
  'mcfed.mapreduce.jdbc.input.orderby'='c_int'
)
;
```

 **Note** The preceding command is for reference only.

17.2.3.2.2. Write and query data

After an external table is created, you can use it in the same way you would use a MaxCompute table. You can execute the **INSERT OVERWRITE | INTO** and **SELECT** statements to write data and query whether the write operation is successful respectively. For more information about the statements, see *DML statements in MaxCompute SQL*.

17.2.3.3. Read data from AnalyticDB

Run the following commands to read data from AnalyticDB:

```
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}'
TBLPROPERTIES(
  'mcfed.mapreduce.jdbc.input.orderby'='c_int'
)
;
-- Create an external table.
select * from ads_read;
-- Query and read data.
```

 **Note** The preceding commands are for reference only.

17.2.4. RDS data source

17.2.4.1. Overview

RDS updates or processes data. If both the data processed by RDS and the data in MaxCompute is used in computation, the data in RDS must be synchronized to MaxCompute. In this case, you can access the data in RDS by creating an external table.

The following examples show how MaxCompute accesses and processes RDS data.

 **Note** When you create an external table, the corresponding table may not exist in RDS. However, when you perform the SELECT or INSERT operation on external tables, you must create the corresponding tables in RDS first.

17.2.4.2. Write data to RDS

17.2.4.2.1. Create an external table

Run the following command to create an external table:

```
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}'
TBLPROPERTIES (
    'mcfed.mapreduce.jdbc.input.orderby'='c_int'
)
;
```

 **Note** The preceding command is for reference only.

17.2.4.2.2. Write and query data

After an external table is created, you can use it in the same way you use a MaxCompute table. You can execute the INSERT OVERWRITE | INTO and SELECT statements respectively to write data and query whether the write operation is successful. For more information about the INSERT OVERWRITE | INTO and SELECT statements, see *DML statements in MaxCompute SQL*.

17.2.4.3. Read data from RDS

Run the following commands to read data from 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}'
TBLPROPERTIES(
  'mcfed.mapreduce.jdbc.input.orderby'='c_int'
)
;
-- Create an external table.
select * from rds_read;
-- Query and read data.
```

 **Note** The preceding commands are for reference only.

17.2.5. HDFS data source (Alibaba Cloud)

17.2.5.1. Overview

Alibaba Cloud Hadoop Distributed File System (HDFS) is a distributed file system designed for Alibaba Cloud computing resources such as ECS and Container Service.

HDFS allows you to manage and access data in the same way as its open-source counterpart. HDFS features such as unlimited capacity, performance scale-out, single namespace, multi-tenancy, high reliability, and high availability, can be used without the need to modify existing big data analysis applications.

MaxCompute can interact with HDFS to jointly compute external tables.

HDFS supports multiple file formats, such as **text file**, **sequence file**, **RC file**, **Parquet**, and **AVRO**. The following example use **text file** to show how MaxCompute accesses and processes HDFS data.

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

Run the following command to create an external table:


```
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 as MountPointId.
-- /user/textfile is the file path. Replace it with the actual file path.
TBLPROPERTIES (
  "mcfed.fs.dfs.impl"="com.alibaba.dfs.DistributedFileSystem"
);
```

 **Note** The preceding command is for reference only.

17.2.5.2.1.2. Write and query data

After an external table is created, you can use it in the same way you use a MaxCompute table. You can execute the **INSERT OVERWRITE | INTO** and **SELECT** statements respectively to write data and query whether the write operation is successful. For more information about the **INSERT OVERWRITE | INTO** and **SELECT** statements, see *DML statements in MaxCompute SQL*.

17.2.5.2.2. Read data from HDFS

Run the following command to read data from HDFS after you upload the textfile file to HDFS:

```
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 as MountPointId.
-- /user/textfile is the file path. Replace it with the actual file path.
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.
```

 **Note** The preceding command is for reference only.

17.2.5.3. Data processing for partitioned tables

Run the following commands to create an external table and process its 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 must be set as MountPointId.
-- user/partition/textfile/ is the file path. Replace it with the actual file path.
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.
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.

```

 **Note** The preceding commands are for reference only.

17.2.6. TDDL data source

17.2.6.1. Overview

By encapsulating MySQL, TDDL provides features such as data partitioning, read/write splitting, and failover. In most cases, TDDL can be directly used to access MySQL databases. TDDL also provides Corona connection mode. Corona is a MySQL proxy that follows the standard MySQL protocol and can use JDBC to establish a connection.

MaxCompute can access MySQL databases of TDDL. Built-in StorageHandlers encapsulate native APIs provided by Hadoop such as org, Apache, Hadoop, MapReduce, lib, and db. MySQL JDBC is used for underlying data communication.

The following topics describe how MaxCompute accesses and processes TDDL data.

 **Notice** Among create, read, update, and delete (CRUD) operations, only the following operations are supported:

- MaxCompute reads data from the external table created for a MySQL database.
- MaxCompute writes data to the external table in the append mode.

17.2.6.2. Prerequisites

Because many features are disabled in the MaxCompute 2.0 by default, you must manually configure the following settings:

```
set odps.sql.hive.compatible=true;
-- You must configure this item for all DDL and DML statements to be used on TDDL external tables.
```

```
set odps.sql.udf.java.retain.legacy=false;
-- You must configure this item for all DDL and DML statements to be used on TDDL external tables.
```

```
set odps.sql.jdbc.splits.num=3;
-- Set the number of splits that MaxCompute reads from the MySQL database. Maximum value: 256. Default value: 1. You must configure this item for the SELECT operation on TDDL external tables.
```

```
set odps.sql.jdbc.reducer.num=3;
-- Set the number of concurrent instances that MaxCompute writes to the MySQL database. Maximum value: 256. Default value: 64. If the number of concurrent instances in the generated execution plan is smaller than this value, no changes are made. You must configure this item for the INSERT operation on TDDL external tables.
```

```
set odps.sql.hive.compatible=true;
-- Use an open-source community API to obtain and parse MySQL data types. You must configure this item for all DDL and DML statements to be used on TDDL external tables.
```

```
set odps.sql.type.system.odps2=true;
-- You must configure this item if new data types TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, and BINARY are involved in SQL operations such as CREATE, SELECT, and INSERT.
```

17.2.6.3. Create a TDDL external table

17.2.6.3.1. Syntax

External tables can act as interfaces between MaxCompute and databases. The method used to process MySQL unstructured data in TDDL is similar to the method to access and process OSS unstructured data. First, you must execute the CREATE EXTERNAL TABLE statement to create an external table. The syntax is as follows:

```
-- Remember to add the corresponding SET statement.
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 (
    ...
);
```

Description:

- **column schema:** supports the following data types.

Data type mapping

MySQL type	MaxCompute type
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 overflow occurs.)

 **Notice** Because unsigned data types are not supported in MaxCompute, loss of precision may occur if unsigned types are specified.

- `setproject odps.sql.udf.strict.mode=true;` (strict mode, which is the default mode).
 - When reading external tables: MaxCompute can read the data if unsigned data is converted to signed data without loss of precision. A RuntimeException ("value out of range") error is reported if loss of precision occurs during data type conversion.
 - When writing external tables: MaxCompute does not check data types. You can specify the SQL mode to let MySQL produce desired data check actions. For more information about SQL mode settings and data check actions, see [Server SQL Modes](#).
- `setproject odps.sql.udf.strict.mode=false;` (non-strict mode)
 - When reading external tables: MaxCompute can read unsigned data that has been converted to signed data without loss of precision. NULL is obtained if loss of precision occurs during data type conversion.
 - When writing external tables: MaxCompute does not check data types. You can specify the SQL mode to let MySQL produce desired data check actions.

- **STORED BY:** Only built-in StorageHandlers are supported. TDDL table field types must be within the range of supported data types in **column schema**.
- **LOCATION:** Three LOCATION formats are supported.

1. Access a MySQL database through a JDBC connection string.

```
jdbc:mysql://<user>:<password>@<host>/<databaseName>? useSSL=false&table=<tableName>
```

user and password are the username and password of the JDBC connection string. host is the network address of the MySQL database. databaseName is the name of the MySQL database. tableName is the name of the MySQL table corresponding to the external table.

2. Access the MySQL database of TDDL through Corona.

```
jdbc:mysql://<user>:<password>@<host>/<databaseName>? useSSL=false&table=<tableName>
```

3. Access the MySQL database of TDDL through an application name.

```
jdbc:mysql://dummy_host? table=<tableName>
```

tableName is the name of the MySQL table corresponding to the external table. You must specify `odps.federation.jdbc.tddl.appname` in the TBLPROPERTIES clause.

 **Notice**

In the first location format, MaxCompute interacts with the database through JDBC. You must enter your username and password as plaintext data, which makes this location format less secure than others. Although usernames and passwords will be hidden when LogView or DESC EXTENDED TABLE is used in MaxCompute, we recommend that you use a separate DDL statement to create an external table before using the external table.

For example, a project member with higher permissions can create an external table in MaxCompute. Other project members can then directly use the external table. This prevents project members with lower permissions from using the plaintext username and password, and prevents the plaintext password from being contained in SQL scripts.

- **TBLPROPERTIES**: includes the following items.

- **odps.federation.jdbc.condition**: specifies the filter when MaxCompute reads data from a MySQL database. The difference between `odps.federation.jdbc.condition` and `select * from text_test_jdbc_write_external where condition` :

Suppose the MySQL table contains 100 rows of data and you want to filter the data such that you obtain 10 rows. When you execute `odps.federation.jdbc.condition`, the MySQL table is filtered and MaxCompute only reads 10 rows from the external table. When you execute `select * from text_test_jdbc_write_external where condition`, MaxCompute reads 100 rows from the MySQL table, and then obtains 10 rows.

- **odps.federation.jdbc.colmapping**: specified 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 into the MySQL database. The following data insertion types are supported: `simpleInsert`, `insertOnDuplicateKeyUpdate`, and `replaceInto`. By default, the insertion type is `simpleInsert` if this parameter is not specified.

The INSERT statement executed in MaxCompute is parsed into the following SQL statements to update 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 value, MaxCompute uses the application name to access the MySQL database in TDDL SDK mode.

17.2.6.3.2. Example

The following example shows how to use the application name to access the MySQL database of TDDL. In this example, the application name is `ODPS_TDDL_TEST_APP` and the table name is `odps_federation_localrun_write`.

Example:

```
-- Remember to add the corresponding SET statement.
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

For complex operations such as GROUP JOIN, we recommend that you import data from external table to MaxCompute tables before performing operations. This improves the efficiency of data computation. 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:

```
-- Remember to add the SET statement.
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, only a data channel is established between MaxCompute and MySQL. MaxCompute does not store any MySQL data. If external table data is lost from the MySQL database, it will not be available in MaxCompute.

When data is imported to a MaxCompute table, the data is actually stored in the MySQL database. If imported data is lost from the MySQL database, it can be retrieved from the MaxCompute table.

17.2.6.5. Write data to an external table in the append mode

The column names and data types of the external table must be consistent with those of the database to ensure that the correct data is written to the external table. For more information about data check actions when loss of precision occurs during data type conversion, see the column schema parameter of [Syntax](#).

An example of the command used is as follows:

```
-- Remember to add the SET statement.
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`. No matter which statement is executed, data is appended to the table and you can use `ODPS.federation.jdbc.insert.type` to specify the data insertion type. For more information, see the `TBLPROPERTIES` parameter in [Syntax](#). However, the preceding syntax notes are not applicable to MaxCompute tables.

17.3. External data sources

17.3.1. HDFS data source (open-source)

17.3.1.1. Overview

HDFS is the most widely used storage service in the open-source community. Most customers use HDFS at the underlying layer of their self-developed big data systems.

MaxCompute uses external tables to access HDFS data to facilitate data migration, interact with self-developed customer systems, and reduce the efforts and costs of customers.

HDFS supports multiple file formats, such as text file, sequence file, RC file, Parquet, and AVRO. The following example use text file to show how MaxCompute accesses and processes HDFS data.

17.3.1.2. Write data to HDFS

17.3.1.2.1. Create an external table

Run the following command to create an external table after the testfile script has been compiled:

```
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/';
-- File path /user/wbyy/ is for reference only. Replace it with the path to actually be accessed.
```

 **Note** The preceding command is for reference only.

17.3.1.2.2. Write and query data

After an external table is created, you can use it in the same way you would use a MaxCompute table. You can execute the `INSERT OVERWRITE | INTO` and `SELECT` statements to write data and check whether the write operation is successful respectively. For more information about the statements, see *DML statements in MaxCompute SQL*.

17.3.1.3. Read data from HDFS

Run the following command to read data from HDFS after you compile the testfile script:

```
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/';
-- File path /user/wbyy/ is for reference only. Replace it with the path to actually be accessed.
-- Create an external table.
select * from testfile_read;
-- Query and read data.
```

 **Note** The preceding command is for reference only.

17.3.2. MongoDB data source

17.3.2.1. Overview

ApsaraDB for MongoDB is a stable, reliable, and auto-scaling database service that is fully compatible with MongoDB protocols. MongoDB offers a full range of database solutions, such as disaster recovery, backup, restoration, monitoring, and alerting.

MaxCompute can interact with MongoDB for joint computation after you create external tables.

The following examples show how MaxCompute accesses and processes MongoDB data.

17.3.2.2. Prerequisites

You must first deploy MongoDB before creating an external table and processing MongoDB data.

1. Run the following command to enable the MongoDB service:

```
bin/mongod --dbpath=./db
```

2. Run the following command to start the 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 operation is successful. A response of 1 indicates a successful operation.

```
db.auth('${user}', '${password}')
```

17.3.2.3. Write data to MongoDB

17.3.2.3.1. Create an external table

Run the following command to create a collection in 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 values to actually be used.
```

After the collection has been created, run the following command 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.${tablename}"
TBLPROPERTIES (
    "mcfed.mongo.input.split_size"="2",
    -- input.split_size value is for reference only. Replace them with the values to actually be used.
    "mcfed.location"="mongodb://${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}"
);
```

 **Note** The preceding commands are for reference only.

17.3.2.3.2. Write and query data

After an external table is created, you can use it in the same way that you would use a MaxCompute table. You can execute the **INSERT OVERWRITE | INTO** and **SELECT** statements to write data and check whether the write operation is successful respectively. For more information about the statements, see *DML statements in MaxCompute SQL*.

17.3.2.4. Read data from MongoDB

Run the following command to read data from MongoDB after a row of data has been inserted into a created collection:

```
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.${tablename}"
TBLPROPERTIES (
    "mcfed.mongo.input.split_size"="2",
    -- The value of the input.split_size parameter is for reference only. Replace it with the value to actually be used.
    "mcfed.location"="mongodb://${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.
```

 **Note** The preceding command is for reference only.

17.3.3. HBase data source

17.3.3.1. Overview

ApsaraDB for HBase is a distributed database based on Hadoop. It can store PBs of data and be used in scenarios requiring high-throughput random read/writes.

MaxCompute can interact with HBase for joint computation after you create external tables.

The following examples show how MaxCompute accesses and processes HBase data.

17.3.3.2. Write data to HBase

17.3.3.2.1. Create an external table

Run the following command to create an external table:

```
set odps.sql.hive.compatible=true;
drop table if exists hbase_table_external;
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'='${table.name}', 'mcfed.hbase.columns.mapping'
  '=':key,cf:a')
-- cf is the column family in the HBase table.
location 'hbase://host:port'
TBLPROPERTIES ('hbase.table.name'='${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 the values to actually be used.
```

 **Note** The preceding command is for reference only.

17.3.3.2.2. Write and query data

After an external table is created, you can use it in the same way that you would use a MaxCompute table. You can execute the **INSERT OVERWRITE | INTO** and **SELECT** statements to write data and check whether the write operation is successful respectively. For more information about the statements, see *DML statements in MaxCompute SQL*.

17.3.3.3. Read data from HBase

Run the following commands to read data from HBase after you have created a table in the HBase client and inserted data into it:

```
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'='${table.name}', 'mcfed.hbase.columns.mapping'
=':key,f1:name,f1:a')
-- f1 is the column family in the HBase table.
location 'hbase://host:port'
TBLPROPERTIES ('hbase.table.name'='${table.name}', 'hbase.columns.mapping'=':key,f1:name,f1: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 the values to actually be used.
-- 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.
```

 **Note** The preceding commands are for reference only.

18. Unstructured data access and processing (inside MaxCompute)

18.1. Overview

MaxCompute has the following problems when processing unstructured data: MaxCompute stores data as volumes and must export generated unstructured data to an external system for processing.

To alleviate these problems, MaxCompute uses external tables to enable connections between MaxCompute and various data types. MaxCompute uses external tables to read and write data volumes as well as process unstructured data from external sources such as OSS.

The following topics describe how MaxCompute accesses and processes volume unstructured data through external tables.

18.2. Create a volume external table

18.2.1. Syntax

You must execute the CREATE EXTERNAL TABLE statement to create an external table.

```
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>']
;
```

Description:

- **STORED BY:** Two built-in StorageHandlers `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 StorageHandlers cannot be used for some reason, you can build a custom StorageHandler.
- **WITH SERDEPROPERTIES:** specifies table attributes such as delimiters for a custom StorageHandler.
- **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 itself.
- The location of a partitioned table must point to the volume itself.
- 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 the StorageHandler resource. To use a custom StorageHandler, you must first export the custom StorageHandler as a JAR package and then add it to MaxCompute as a JAR resource.

18.2.2. Use the built-in StorageHandler to create an external table

You can use the built-in StorageHandler to create a partitioned 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'--The built-in StorageHandler.
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'--The built-in StorageHandler.
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

When the built-in StorageHandlers are unable to meet the requirements of your business, you can customize a StorageHandler through Java and specify some attributes of the Volume external table which you want to process data through.

The following example shows how to use a custom StorageHandler to create an external table.

Assume that the data type is TEXT and the column delimiter is "|". You can perform the following steps to create an external table:

1. Use MaxCompute Studio or MaxCompute Eclipse development plug-in to customize various 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 operate the volume data through the external table.

18.3. Access a volume external table

Volume external tables can be accessed in the same way that you would access a MaxCompute table.

Example:

```
select key,value from volume_ext_pt where ds="20190102";
```

19. Multi-region cluster deployment on MaxCompute

19.1. Overview

MaxCompute supports multi-region cluster deployment.

The control clusters is deployed in a centralized manner and configures resources as well as manages compute tasks. The Compute cluster is deployed independently for each region and handles project creation and compute task delegation.

19.2. Characteristics of multi-region deployment

This topic describes the characteristics of multi-region deployment.

The multi-region deployment on MaxCompute has the following features:

- One MaxCompute service instance can manage multiple clusters that are located in different cities.
- Cross-cluster data interaction is implemented within the MaxCompute service, and cross-cluster data replication and synchronization are managed based on the actual configuration.
- Metadata is stored in a centralized manner. Therefore, the infrastructure requirements, such as network connections between different data centers, are relatively high.
- A unified account system is required.
- Big data application development systems, such as DataWorks, are used in all regions and clusters.
- You must change the mode of the current MaxCompute cluster to cross-cluster for multi-region deployment.

-  **Note** The conditions and limits on changing the cluster mode are as follows:
- The network bandwidth must be sufficient for multi-region data synchronization and link redundancy.
 - The central-region control cluster has a relatively high latency for basic services such as DNS and TableStore. We recommend that you deploy basic services in the same data center to keep the network latency within 5 ms.
 - The network latency between the central-region control cluster and computing clusters in other regions must be kept within 20 ms.
 - Clocks must be synchronized between clusters in different cities and between machines in the same cluster.
 - The network bandwidth must be sufficient for data replication among clusters.
 - DNS is required.
 - In order for cross-cluster servers to communicate on the same network, the clusters must have similar network bandwidth conditions, such as GE or 10GE.

- The operations, maintenance, deployment, and upgrade for multi-region deployment are different from those for a single cluster deployment. Multi-region deployment requires higher onsite

operations and maintenance capabilities.

19.3. Instructions on multi-region deployment

This topic describes multi-region deployment.

The instructions to implement multi-region deployment are as follows:

- Computing tasks on MaxCompute are implemented in a cluster based on data distribution within the service. Cross-cluster replication and direct reads are performed based on the cluster configuration and distribution of data.
- Data replication and synchronization can be performed between different clusters by table or partition based on the actual cluster configuration.
- You must plan the relationship between the MaxCompute project and the cluster at the business layer. You can select one or multiple clusters for a single project.

 **Note** If you select multiple clusters for a project, data replication is carried out among these clusters.

- If the project contains business that cannot implement cross-cluster computing, you can distribute data to different clusters and divide the computing tasks based on their distribution.

 **Note** Cross-cluster data replication and management requires management policies.

- The bandwidth between clusters in different remote data centers must be sufficient for data replication or read-only operations.
- Replication and read-only operations across clusters in MaxCompute involving large amounts of data will consume all of the bandwidth between the two clusters, as the two data centers share bandwidth.
- A cluster failure in the primary data center will affect the entire service because the primary data center stores information such as metadata and accounts.
- When the secondary data center fails, the project where data is stored on the cluster and the O&M of services are affected.

 **Note** If project data is independently stored in the primary data center, the failure of the secondary data center will not affect computing tasks of the project.

- In general scenarios, data is partitioned 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.

19.4. Multi-region deployment examples

19.4.1. Synchronize table data among multiple clusters

This topic provides an example on how to synchronize table data among multiple 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 multiregion, and AT-MDU-TEST and AT-5KN are selected for it, with AT-MDU-TEST as the default cluster.
- The URL of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`, which is port 9090 of MaxCompute AG.
- The operation entry to create the project is **MaxCompute Configuration > Project Management > Create Project**.

- You have configured cross-cluster replication for the project. The following example demonstrates how to configure it.
 - Choose **MaxCompute Configuration > Global Cross-cluster Replication Configuration** to go to the **Global Cross-cluster Replication Configuration** page.
 - In the Add Item part of the **InfoBetweenClusters** section, set key to **AT-5KN#AT-MDU-TEST**, set both availableBandwidth and totalBandwidth to **2000**, and click **add**.
 - Choose **MaxCompute Configuration > Project Management**. Find multiregion and click **Cross-cluster Replication Configuration**.
 - In the dialog box that appears, configure the parameters, as shown in the following figure.
 - Click **Save** and then click **Start Replication**.

Procedure

Example:

- 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`.
- 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;
```

- Run the following command to check whether the data is inserted into the table:

```
select * from t1;
```

- 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 both steps 4 and 5, the cross-cluster data synchronization operation is successful.

19.4.2. Query the status of data synchronization between primary and secondary clusters

This topic provides an example on 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 multiregion, and AT-MDU-TEST and AT-5KN are selected for it, with AT-MDU-TEST as the default cluster.
- The URL of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`, which is port 9090 of MaxCompute AG.
- The operation entry to create the project is **MaxCompute Configuration > Project Management > Create Project**.

- You have configured cross-cluster replication for the project. The following example demonstrates 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. Compile the bodychecksync configuration file, which is used to send admintask.

```

<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 admintask:

```
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 20180711050550317gnege3ms 2;"
```

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.

19.4.3. 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.

 **Note** When cross-cluster access is not configured and the data volume is large, 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 testmdu, with AT-MDU-TEST as the default cluster. AT-5KN is selected for testcross5kn and serves as the default cluster.
- The URL of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`, which is port 9090 of MaxCompute AG.
- The operation entry to create the projects is **MaxCompute Configuration > Project Management > Create Project**.

2. Create tables for testmdu and testcross5kn.

```
create table testrep(s string );
create table tablecross5kn (s string );
```

3. Construct some data for tables testrep and tablecross5kn.
4. Run the following commands to read the table data in testmdu directly from testcross5kn:

```
use testcross5kn;
select * from testmdu.testrep;
```

 **Note** If the specified data is returned, the cross-region direct read operation 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 testmdu, with AT-MDU-TEST as the default cluster. AT-5KN is selected for testcross5kn and serves as the default cluster.
- The URL of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`, which is port 9090 of MaxCompute AG.
- The operation entry to create the projects is **MaxCompute Configuration > Project Management > Create Project**.

2. Create tables for testmdu and testcross5kn.

```
create table testrep(s string );
create table tablecross5kn (s string );
```

- Construct some data for tables testrep and tablecross5kn.
- Configure cross-cluster replication for testmdu.

 **Note** Choose **MaxCompute Configuration > Project Management**. Find testmdu and click **Cross-cluster Replication Configuration**.

- Run the following commands to read the table data in testmdu directly from testcross5kn:

```
use testcross5kn;
select * from testmdu.testrep;
```

 **Note** If the specified data is returned, the cross-region direct read operation is successful.

19.4.4. Cross-region JOIN

This topic provides an example of cross-region JOIN in multi-region deployment scenarios.

Example:

- In the AdminConsole, create two projects: crosssregion and crossregion02.

 **Note**

- In this example, AT-MDU-TEST and AT-70N are selected for crosssregion, with AT-MDU-TEST as the default cluster. AT-70N is selected for crossregion02 and serves as the default cluster.
- The URL of Apsara Stack MaxCompute AdminConsole is `http://{odps_ag}:9090`, which is port 9090 of MaxCompute AG.
- The operation entry to create the projects is **MaxCompute Configuration > Project Management > Create Project**.

- Configure cross-cluster replication for crosssregion.

 **Note** Choose **MaxCompute Configuration > Project Management**. Find crosssregion and click **Cross-cluster Replication Configuration**.

- Create a table named business for crosssregion.

```
create table business (bid bigint, name string, phone string, address string, region string)
;
```

- Construct some data for the table.
- Run the following command to import data into the table:

```
tunnel upload business business;
```

6. Create a table named product for crossregion02.

```
create table if not exists product(pid bigint,name string,type string,color string,bid bigint);
```

7. Construct some data for the table.
8. Run the following command to import data into the table:

```
tunnel upload product product;
```

9. Run the following command in the console to obtain specific data from the preceding two tables:

```
select pro.pid,pro.name,pro.type,pro.color,bus.name,bus.phone from crossregion02.product pro join crosssregion.business bus on pro.bid=bus.bid where bus.region='Shanghai';
```

 **Note** If the specified data is returned, the cross-region JOIN operation is successful.

20. Security solution

20.1. Target users

This User Guide is intended for all owners and administrators of MaxCompute projects, and users interested in the MaxCompute multi-tenant data security system. The MaxCompute multi-tenant data security system includes:

- User authentication
- User and authorization management of projects
- Cross-project resource sharing
- Project protection

20.2. Quick start

Add a user and grant permissions to the user

Scenario: Jack is the administrator of the prj1 project. A new team member Alice, who already has an Alibaba Cloud account (alice@aliyun.com), applies to join the project. Alice applies for the following permissions: viewing table lists, submitting jobs, and creating tables.

The admin performs the following operations to add Alice to the project:

```
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.
```

Add a user and grant permissions to the user using an ACL

Scenario: Jack is the administrator of prj1. Three new members Alice, Bob, and Charlie join in as data reviewers. They require the following permissions: viewing table lists, submitting jobs, and reading the table userprofile.

The project administrator can use object-based ACL authorization in this scenario.

The operations are as follows:

```
use prj1
add user aliyun$alice@aliyun.com
-- Add a user.
add user aliyun$bob@aliyun.com
add user aliyun$charlie@aliyun.com
create role tableviewer
-- Create a role.
grant List, CreateInstance on project prj1 to role tableviewer; --Grant permissions to the
role
-- Grant permissions to the role.
grant Describe, Select on table userprofile to role tableviewer
grant tableviewer to aliyun$alice@aliyun.com
-- Grant the tableviewer role to a user.
grant tableviewer to aliyun$bob@aliyun.com
grant tableviewer to aliyun$charlie@aliyun.com
```

Package and share resources

Scenario: Jack is the administrator of prj1. John is the administrator of prj2. Due to business requirements, Jack wants to share some resources of prj1 (such as datamining.jar and sampletable) to John's prj2. A user in prj2 (Bob) requires access to these resources. The prj2 administrator can configure an ACL or policy to automatically authorize prj2 users to access these resources, without the intervention of Jack.

The operations are as follows:

1. Prj1 administrator Jack creates a resource package in prj1.

```
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 the table to the package.
allow project prj2 to install package datamining
-- Share the package to prj2.
```

2. Prj2 administrator Bob installs the package in prj2.

```
use prj2
install package prj1.datamining
-- Install the package.
describe package prj1.datamining
-- View the resource list of the package.
```

3. Configure automatic authorization for Bob on the package.

```
use prj2
grant Read on package prj1.datamining to user aliyun$bob@aliyun.com
-- Use an ACL to allow Bob to use the package.
```

 **Note** For more information about cross-project resource sharing, see [Cross-project resource sharing](#).

Configure project protection

Scenario: Jack is the administrator of project prj1. This project contains sensitive data such as user IDs and shopping records. The project also stores many data mining algorithms to which the organization holds intellectual property rights. Jack wants to protect the sensitive data and algorithms in the project. He wants the data to be accessible only to users in the project. The data must not be able to flow out of the project.

The operations are as follows:

```
use prj1
set ProjectProtection=true
-- Enable project protection.
```

When project protection is enabled, data in the project can flow only within the project. Data cannot flow out. In some cases, for example, a user (Alice) requires to export data tables for business purposes. This operation is approved by the project administrator. MaxCompute provides two methods to export data from a protected project.

Method 1: Create an exception policy. For more information, see [Data export methods when project protection is enabled](#).

1. Create a policy file. Create a policy file named `/tmp/exception_policy.txt`. It only allows Alice to export t1 from prj1 using a SQL task. The policy is defined as follows:

```
{
  "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. Configure the exception policy.

```
use prj1
-- Enable project protection and configure an exception policy.
set ProjectProtection=true with exception /tmp/exception_policy.txt
```

 **Note** When you configure the exception policy, ensure that the principal cannot update the data resources or recreate an object with the same name (using DROP TABLE and CREATE TABLE). This prevents data leakage due to time-of-check to time-of-use (TOC2TOU).

Method 2: Configure trusted projects. Configure prj2 as a trusted project of prj1 to enable data flow from prj1 to prj2. For more information, see [Data export methods when project protection is enabled](#).

```
use prj1
add trustedproject prj2
```

 **Note** In MaxCompute, package-based resource sharing and project protection are mutually independent mechanisms that take effect at the same time, but their functions are mutually restrictive.

In MaxCompute, resource sharing has a higher priority than project protection. This means, if an object in a protected project is shared with other projects through the package mechanism, cross-project access to this object is not subject to the project protection rules.

20.3. User authentication

The main purpose of user authentication is to verify the identity of a request sender. Authentication typically includes:

- Verifying the true identity of a message sender
- Checking whether the message was tampered with before it is received.

20.4. Project user and authorization management

20.4.1. Overview

Projects are the foundation of the MaxCompute multi-tenant system and the basic units of data management and computing. When you create a project, you are automatically the project owner. All objects in the project, such as tables, instances, resources, and UDFs, belong to you. Objects in the project can only be accessed by the owner and users that are authorized by the owner.

This topic describes users, roles, and authorization management of projects. For example, Alice is the owner of test_project, and another user from Alice's project team requests to access the resources in test_project. Alice can use the methods described in this topic to perform user and authorization management. If a user that wants access to Alice's project is not from her project team, Alice can implement cross-project sharing. For more information, see [Cross-project resource sharing](#).

20.4.2. User management

Add a user

If Alice (the project owner) decides to authorize another user, she must add the user to this project. Only users in a project can be authorized.

Run the following command to add a user:

```
add user <full_username>
-- Add a user to a project.
```

Remove a user

When a user leaves the project team, Alice needs to remove the user from the project. After a user is removed from the project, the user no longer has any access permissions on project resources.

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 any access permissions on project resources.
- 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 authorization related to the user is retained. However, the policy authorization at the role level is revoked, and the policy authorization at the project level is retained. If the user is added to the project again, the previous ACL authorization of the user is re-activated.
- MaxCompute does not support complete removal of a user and the relevant authorization data.

20.4.3. Role management

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. When granting permissions to users, you should consider using role-based authorization.

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 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.

The role management commands are as follows:

```
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 the role of a user.
```

 **Note** When you delete a role, MaxCompute checks whether there are users assigned with this role. If the role is assigned to users, the role fails to be deleted. To delete the role, you must revoke this role from all users.

20.4.4. ACL authorization actions

Authorization usually involves three elements: subject, object, and action. In MaxCompute, a subject is the user, there are various types of objects in a project, and actions are performed on objects. Different types of objects support different actions.

MaxCompute projects support the following object types and actions:

Object types and actions

Object	Action	Description
Project	Read	Check the information about the project itself (not including any objects of the project), such as CreateTime.
Project	Write	Update the information of the project itself (not including any objects of the project), such as Comments.
Project	List	View a list of all types of objects in the project.
Project	CreateTable	Create a table in the project.
Project	CreateInstance	Create an instance in the project.
Project	CreateFunction	Create a function in the project.
Project	CreateResource	Create resource in the project.
Project	CreateJob	Create a job in the project.
Project	CreateVolume	Create a volume in the project.
Project	All	All the permissions above.
Table	Describe	Read the metadata of the table.
Table	Select	Read the information of the table.
Table	Alter	Alter the metadata of the table; add or drop partitions.
Table	Update	Override or add data to the table.
Table	Drop	Drop the table.
Table	All	All the preceding permissions.
Function	Read	Read and execute permissions.
Function	Write	Update.
Function	Delete	Delete.
Function	All	All the preceding permissions.
Resource, instance, job, volume	Read	Read permissions.

Object	Action	Description
Resource, instance, job, volume	Write	Update permissions.
Resource, instance, job, volume	Delete	Delete permissions.
Resource, instance, job, volume	All	All the preceding permissions.

 **Note** In the preceding permissions, the CreateTable action of project objects, as well as the Select, Alter, Update, and Drop actions of table objects, must be used together with the CreateInstance action of project objects. Before using the preceding permissions to complete actions, you must assign the CreateInstance permission.

After adding users or creating roles, these users or roles should be authorized. The ACL authorization mechanism of MaxCompute is object-based. Authorization data (the access control list, or ACL) is considered as a sub-resource of an object). Therefore, ACL authorization can be performed only when the objects exist. When the objects are deleted, authorized permission data is automatically deleted.

The ACL of MaxCompute supports authorization using commands like SQL92-defined GRANT/REVOKE commands. Use the corresponding authorization commands to authorize existing project objects or revoke their authorization.

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 ACL authorization commands of MaxCompute do not support the [WITH GRANT OPTION] parameter. That is, when 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 completed by users with at least one of the following identities:

- Project owner
- Users with the admin role in the project
- Object creators in the project

ACL authorization example:

Scenario: Users `alice@aliyun.com` and `bob@aliyun.com` are new members of `test_project`. In `test_project`, they must submit jobs, create data tables, and view existing objects of the project. The administrator then takes the following authorization actions:

```
use test_project
-- Open a project.
security
add user aliyun$alice@aliyun.com
-- Add a user.
add user aliyun$bob@aliyun.com
-- Add a user.
create role worker
-- Create a role.
grant worker TO aliyun$alice@aliyun.com
-- Assign a role.
grant worker TO aliyun$bob@aliyun.com
-- Assign a role.
grant CreateInstance, CreateResource, CreateFunction, CreateTable, List ON PROJECT test_project TO ROLE worker
-- Authorize a role.
```

20.4.5. View permissions

MaxCompute allows you to view permissions in different dimensions. For example, you can view permissions of a specified user, permissions of a specified role, or the authorization list of a specified object.

View the permissions of a user

```
show grants
-- View the access permissions of the current user.
show grants for <username>
-- View the access permissions of a specified user. Only project owners and administrators can view the access permissions of a specified user.
```

View the permissions of a role

```
describe role <rolename>
-- View the access permissions granted to a specified role.
```

View the authorization list of an object

```
show acl for <objectName> [on type <objectType>]
-- View the authorization list of a specified object.
```

 **Note** If `[on type <objectType>]` is not specified, the default type is `table`.

MaxCompute uses characters A, C, D, and G to indicate the permissions of users or roles. The characters are described as follows:

- A: allow. Access is allowed.

- D: deny. Access is denied.
- C: condition. This is a conditional authorization. This character appears only in the policy authorization system. For more information, see [Condition block structure](#).
- G: grant. You can grant permissions to this object.

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
```

20.5. Cross-project resource sharing

20.5.1. Overview

You are the owner or administrator (admin role) of a project, and someone requests to access resources of your project. If the applicant is a member of your project team, we recommend that you use the user and authorization management features for your project. For more information, see [User and authorization management of projects](#). If the applicant is not a member of your project team, you can use the package-based resource sharing feature described in this topic.

A package is used to share data and resources across projects. It can be used to implement cross-project user authorization. The following scenario describes a problem that can only be resolved effectively with the package mechanism.

Members of the Alifinance project need to access Alipay project data. The Alipay project administrator adds Alifinance project users to the Alipay project, and then grants the new users common permissions. For security concerns, the Alipay project administrator does not want to authorize every user of the Alifinance project team. A mechanism is required to allow the Alifinance project administrator to control access to the authorized objects.

By using the package feature, 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 installing the package, the Alifinance project administrator can decide whether to grant permissions on the package to users 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 corresponding access permissions, and provides the package receiver with the permissions to install and use the package. The package user consumes the resources. After installing 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.

20.5.2. Package usage

20.5.2.1. Operations for package creators

Create a package

Run the following commands to create a package:

```
create package <pkgname>
```

Delete a package

Run the following commands to drop a package:

```
delete package <pkgname>
```

Add a resource to be shared to the package

Run the following commands to add a resource to the 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 re
s_name
privileges ::= action_item1, action_item2, ...
```

Note

- The types of supported objects exclude projects, so you cannot use a package to create objects in other projects.
- In addition to the objects, the operation permissions on the objects are also added to the package. When not passed [with privileges Privileges] When you specify an action permission, the default is read-only, that is, read/describe/select. An object (resource) and its permissions are considered as a whole. You can delete resources in a package. The permissions are revoked when resources are deleted.

Allow other projects to use a package

Run the following commands to allow other projects to use a package:

```
allow project <prjname> to install package <pkgname> [using label <number>]
```

Revoke the permission for other projects to use a package

Run the following commands to revoke another project's permission to use package:

```
disallow project <prjname> to install package <pkgname>
```

View the list of packages already created and installed

Run the following commands to view the list of packages already created and installed:

```
show packages
```

View details of a package

Run the following commands to view details of a package:

```
describe package <pkgname>
```

20.5.2.2. Operations for package users

The installed package is a type of independent object in MaxCompute. To access resources in a package (other projects' resources shared with you), you must have the permission to read the package. If you do not have read permissions, submit an application to the project owner or admin for the permissions. The project owner or admin can grant the permissions by using ACL authorization or policy authorization.

For example, the following ACL authorization rule allows user `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
```

The following policy authorization rule allows any user in `prj2` to access resources in a package:

```
use prj2
install package prj1.testpkg
put policy /tmp/policy.txt
```

The contents of `/tmp/policy.txt` are as follow:

```
{
  "Version": "1", "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "odps:Read", "Resource": "acs:odps:*:projects/prj2/packages/prj1.testpkg"
  }]
}
```

Install a package

Run the following commands to install a package:

```
install package <pkgname>;
```

 **Note** The pkgName of a package to be installed must be in the format of <projectName>. <packageName>.

Uninstall package

Run the following commands to uninstall a package:

```
uninstall package <pkgname>;
```

 **Note** The pkgName of a package to be uninstalled must be in the format of <projectName>. <packageName>.

View packages

Run the following commands to view packages:

```
show packages
-- View the list of packages already created and installed.
describe package <pkgname>
-- View details of a package.
```

20.6. Project protection

20.6.1. Overview

Some enterprises (such as financial institutions and military enterprises) have high data security requirements. For example, their employees can only perform their jobs 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 leakage 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. You are required to perform the following configurations to prohibit all operations that could result in data outflow.

20.6.2. Data protection

MaxCompute provides a project protection mechanism that prohibits operations that introduce data leakage risks. You can simply configure your project as follows to enable project protection:

```
set security.ProjectProtection=true
-- Enable project protection. This rule allows inbound data flows, but prohibits outbound data flows.
```

After project protection is enabled, the data flow of the project is controlled. Data can flow in, but cannot flow out.

Project protection is disabled by default (`ProjectProtection = false`). If you have access permissions on multiple projects, you can use any cross-project data access to migrate data between projects. If a project stores highly-sensitive data, the administrator must configure a project protection mechanism.

20.6.3. Data export methods when project protection is enabled

After you enable project protection, you may soon encounter this situation: A user (Alice) submits a request to export the data of a table from the project. It is verified that this table contains no sensitive data. MaxCompute provides two methods to export data after project protection is enabled.

Configure an exception policy

The project owner can run the following command to attach an exception policy when enabling project protection:

```
SET ProjectProtection=true WITH EXCEPTION <policyFile>
```

 **Note** This policy mechanism is different from the policy-based authorization mechanism (though the command syntax is the same). This policy describes exceptions of the project protection mechanism. All access requests matching the policy are not subject to the project protection rules.

Example:

The following policy allows the user `Alice@aliyun.com` to export data out of the `alipay` project when performing the `SELECT` operation on the `alipay.table_test` table in a SQL task:

```
{
  "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 preceding exception policy does not grant any permissions. If Alice does not have the SELECT permission on alipay.table_test, the preceding exception policy does not allow Alice to export data. Project protection specifies data flow control, not access control. Data flow control is effective only when a user can access the target data.
- Data leakage due to TOC2TOU (also known as the race condition problem) arises in the following situation:
 - i. [TOC stage] User A submits an application to the project owner to export table t1. The project owner verifies that t1 does not contain sensitive data. The project owner configures an exception policy, which allows user A to export t1.
 - ii. A malicious user changes the content of t1 by writing sensitive data to it.
 - iii. [TOU stage] User A exports t1. The t1 exported by the user is not the same t1 that was authorized by the project owner.

Suggestions on TOC2TOU prevention: For a table that a user applies to export, the project owner must make sure that no other user (including admins) can update the table or create a table with the same name (using DROP TABLE and CREATE TABLE). In the preceding TOC2TOU scenario, we recommend that the project owner create a snapshot of t1 in step 1. Then, create an exception policy for the user to use this snapshot. Do not grant the admin role to any users.

Configure a trusted project

If the current project is protected, and the target project is a trusted project of the current project, data flows to the target project are not subject to the project protection rules. If each project in a group is mutually configured as trusted projects, the group is considered a trusted project group. Data can flow freely within the group, but cannot flow out.

Run the following command to manage trusted projects:

```
list trustedprojects
-- Show all trusted projects of the current project.
add trustedproject <projectname>
-- Add a trusted project of the current project.
remove trustedproject <projectname>
-- Remove a trusted project of the current project.
```

20.6.4. Resource sharing and data protection

In MaxCompute, package-based resource sharing and project protection are mutually independent mechanisms that take effect at the same time, but their functions are mutually restrictive.

In MaxCompute, resource sharing takes precedence over project protection. If a data object is shared to users in other projects through resource sharing, the project protection rules will not apply to this data object.

To prevent data outflow from the project, after you enable project protection (ProjectProtection=true), you must verify the following points:

- Make sure that no trusted projects are added. If one is added, evaluate possible risks.

- Make sure that no exception policies are configured. If one is configured, evaluate possible risks, especially risks due to TOCTOU.
- Check whether package data sharing is not in use. If package data sharing is in use, make sure that the package contains no sensitive data.

20.7. Project security configuration

MaxCompute is a multi-tenant data processing platform. Different tenants may have different data security requirements. MaxCompute provides project-level security configuration to satisfy data security requirements of different tenants. Project owners can customize their external accounts and authentication models as required.

MaxCompute supports multiple orthogonal authorization mechanisms, such as ACL-based authorization, policy-based authorization, and implicit authorization (for example, an object creator is automatically authorized to access the object). However, not all users require these security mechanisms. You can configure an authentication model that best suits your business demands and usage habits.

```
show SecurityConfiguration
-- View the security configuration of the project.
set security.CheckPermissionUsingACL=true/false
-- Enable or disable ACL-based authorization. Default value: true.
set security.CheckPermissionUsingPolicy=true/false
-- Enable or disable policy-based authorization. Default value: true.
set security.ObjectCreatorHasAccessPermission=true/false
-- Allow or disallow an object creator to be granted the object access permission by default. Default value: true.
set security.ObjectCreatorHasGrantPermission=true/false
-- Allow or disallow an object creator to be granted the authorization permission by default. Default value: true.
set security.LabelSecurity=true/false
-- Enable or disable the label security policy.
set security.ProjectProtection=true/false
-- Enable or disable project protection to allow or prohibit data transfer from the project.
```

20.8. Authorization policies

20.8.1. Policy overview

Policy authorization is a principal-based authorization. Permission data authorized by policy (that is, access policy) is considered as a type of sub-resource of the authorization subject. Policy authorization can be performed only if the subject exists. When a subject is deleted, their authorization data is deleted automatically. Policy authorization uses an access policy language customized for MaxCompute to allow or deny subjects access to project objects.

Policy authorization is a new authorization mechanism mainly used to handle complicated authorization scenarios that ACL authorization struggles to deal with, such as:

- Authorize a group of objects, such as all functions and all tables starting with taobao, at a time.
- For authorizations with restrictive conditions, such as one that takes effect only in a specified period, one that takes effect only if the requester initiates the request from a specified IP addresses, or one that allows the user to use SQL only (disallowing other tasks) to access a table.

The command format of policy authorization is as follows:

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

 **Note** MaxCompute currently supports project policies and role policies. A project policy applies to all users of the project, while a role policy applies only to users to whom the role is assigned. You must specify a principal (user) for project policies, but you cannot specify a principal for role policies, because the role will be assigned to users.

An example of project policy authorization is as follows:

Scenario: Authorized user `alice@aliyun.com` can only submit a request before 23:59:59 2017-11-11 from an IP address in the subnet 10.32.180.0-23, and can only perform the `CreateInstance`, `CreateTable`, and `List` operations in `test_project`. No tables in `test_project` can be dropped.

The policy is as follows:

```
{
  "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

- Currently, only role policies and project policies are supported, and user policies are not.
- Every policy only supports one policy file. Since put policies override existing policies, you must follow the sequence below to modify a policy:
 - i. Get Policy.
 - ii. Merge policy statements.
 - iii. Put policies.

20.8.2. Policy-related terms

Permission is a basic concept of access control. When 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, and policy refers to a set of statements.

An access policy comprises the following access control elements: principal, action, resource, access restriction, and effect. These elements are briefly described below.

Principal

A principal of an object is a user or group 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, 2017. Michael is the principal of the object.

Action

An action is an activity that the principal has permission to perform. For example, the access policy allows Michael to perform the CreateObject action on the resource SampleBucket before December 31, 2017. Therefore, CreateObject is an action of the access policy.

Resource

Resource is the object a principal requests access to. For example, the access policy allows Michael to perform the CreateObject action on the resource SampleBucket before December 31, 2017. SampleBucket is a resource of the access policy.

Access restriction

Access restriction is the prerequisite for the permission to take effect. For example, the access policy allows Michael to perform the CreateObject action on resource SampleBucket before December 31, 2017. The access restriction is before December 31, 2017.

Effect

Authorization effect has two options: Allow (action) or deny (action). In general, deny actions are generally more efficient and are checked first during permission checks.

 **Notice** The deny action and revoking permission are completely different concepts. The latter usually revokes permissions for both allow action and deny action. For example, a traditional database supports the revoke and revoke deny actions.

20.8.3. Access policy structure

20.8.3.1. Overview

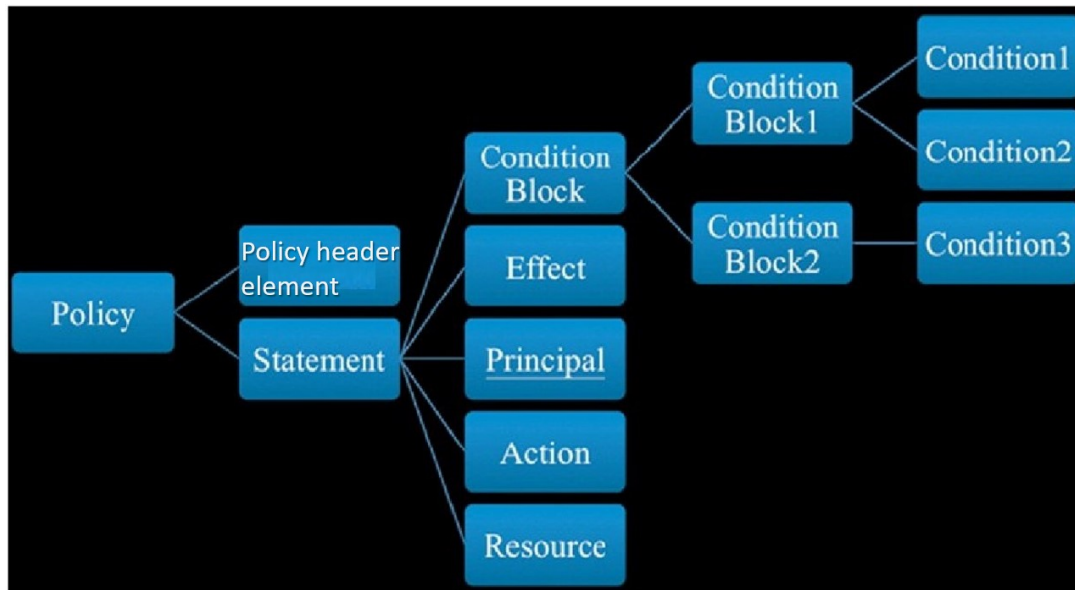
The following figure shows the structure of an access policy. A policy consists of the following parts:

- An optional policy header
- One or more statements

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.

Policy structure



20.8.3.2. Authorization statement structure

An authorization statement includes the following entries:

- **Effect**: indicates the permission type of this statement. The value can be either 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 appoint 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 more operation names, and supports the asterisk (*) and question mark (?) wildcard characters. The asterisk (*) matches any number of characters, and the question mark (?) matches a single character. For example, Action = * indicates all operations.
- **Resource**: indicates the authorization object. It can be one or more object names, and supports the asterisk (*) and question mark (?) wildcard characters. The asterisk (*) matches any number of characters, and the question mark (?) matches a single character. 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. See the next topic for the structure of the condition block.

20.8.3.3. Conditional block structure

A condition block consists of one or more condition clauses. A condition clause consists of an action type, keyword, and condition value. The action types and keywords will be described in detail in the subsequent sections.

Whether a condition block is satisfied is determined as follows:

- A conditional keyword can correspond to one or more values. If the conditional keyword value is equal to one of the corresponding values, the condition is satisfied.
- A condition clause of a conditional operation type is satisfied if all conditional keywords in the clause are satisfied.
- A condition block is satisfied only if all of its condition clauses are satisfied.

20.8.3.4. Conditional action type

The following action types are supported: string, number, date, Boolean, and IP address. The methods supported by each conditional operation type are as follows:

String:

```
StringEquals  
StringNotEquals  
StringEqualsIgnoreCase  
StringNotEqualsIgnoreCase  
StringLike  
StringNotLike
```

Numeric:

```
NumericEquals  
NumericNotEquals  
NumericLessThan  
NumericLessThanEquals  
NumericGreaterThan  
NumericGreaterThanEquals
```

Date and time:

```
DateEquals  
DateNotEquals  
DateLessThan  
DateLessThanEquals  
DateGreaterThan  
DateGreaterThanEquals
```

Boolean:

```
Bool
```

IP address:

```
IpAddress NotIpAddress
```

20.8.3.5. Conditional keywords

MaxCompute supports the conditional keywords reserved by Alibaba Cloud Service (ACS). The following table describes these conditional keywords.

Conditional keywords

Conditional keywords reserved by ACS	Type	Description
acs:CurrentTime	Date and time	The time when the Web server receives a request. It is based on the ISO 8601 standard, for example, 2017-11-11T23:59:59Z.
acs:SecureTransport	Boolean	Whether the request is sent over a secure channel, such as an HTTPS channel.
acs:SourceIp	IP address	The IP address of the client that sent the request.
acs:UserAgent	String	The UserAgent of the client that sent the request.
acs:Referer	String	The HTTP referer that sent the request.

 **Note** acs:SourceIp refers to the remote_ip of the HTTP connection, not the (leftmost) client IP address in the x-forwarded-for HTTP header field. For example, if 10.230.205.105 is a LAN IP address, acs:SourceIp is the egress gateway IP address of this LAN. If the network egress uses a proxy server, acs:SourceIp is the IP address of the proxy server. If the request traverses across multiple proxy servers, acs:SourceIp is the IP address of the final proxy server. The value of acs:SourceIp may vary depending on the rules configured on the proxy server.

20.8.4. Access policy norm

20.8.4.1. Principal naming convention

The principal is the request sender. Currently, only an Alibaba Cloud account, domain account, or Taobao account is accepted as a principal. A 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"]
```

20.8.4.2. Resource naming convention

The following naming conventions are used for MaxCompute resources.

```
acs:<service-name>:<namespace>:<relative-id>
```

The parameters are described as follows.

Parameters

Name	Description
acs	Retained resource header.

Name	Description
service-name	The name of an open cloud service, such as MaxCompute, OSS, and TableStore.
namespace	Naming space, used for resource isolation. If a cloud account ID is used for resource isolation, the value can be the cloud account ID. If this option is not supported, use the asterisk (*) wildcard character instead.
relative-id	Indicates the service-related resource. Its meaning depends on specific services. This part of the format description supports a tree structure similar to the file path. Using MaxCompute as an example, the format of relative-id is: <pre>projects/<project_name>/<object_type>/<object_name></pre>

Some MaxCompute resource naming examples.

Naming examples

Item	Description
*	All objects in the project.
projects/prj1/tables/t1	Table t1 of Project prj1.
projects/prj1/instances/*	All instances of Project prj1.
projects/prj1/tables/*	All tables of Project prj1.
projects/prj1/tables/taobao*	All tables of Project prj1 whose names start with "taobao".

20.8.4.3. Action naming

Action naming conventions are as follows:

```
<service-name>:<action-name>
```

Description:

- **service-name**: name of an open cloud service, for example, maxcompute, oss, and table store.
- **action-name**: name of service-related action APIs.

The following table lists MaxCompute action naming examples.

MaxCompute action naming examples

Naming example	Description
*	All actions.

Naming example	Description
odps:*	All MaxCompute actions.
odps:CreateTable	The CreateTable action of MaxCompute.
odps:Create*	All MaxCompute actions whose names start with "Create".

20.8.4.4. Condition keys naming

The naming format for condition keys retained by the open cloud service is:

```
acs:<condition-key>
```

Description:

condition-key: ACS reserves 5 types of condition keys, which are accessible for all open services. They are: acs:CurrentTime, acs:SecureTransport, acs:SourceIp, acs:UserAgent, acs:Referer.

The naming format for condition keys related to the specific service is:

```
<service-name>:<condition-key>
```

Description:

Condition-key: service-defined condition key.

20.8.4.5. Access policy example

Policy example:

```
{
  "Version": "1",
  "Statement": [{
    "Effect": "Allow",
    "Principal": "ALIYUN$alice@aliyun.com",
    "Action": ["odps:CreateTable", "odps:CreateInstance", "odps:List"],
    "Resource": "acs:odps:*:projects/prj1",
    "Condition": { "DateLessThan": {
      "acs:CurrentTime": "2017-11-11T23:59:59Z"
    }
  },
  "IpAddress": { "acs:SourceIp": "10.32.180.0/23"
  }
  },
  {
    "Effect": "Deny", "Principal": "ALIYUN$alice@aliyun.com",
    "Action": "odps:Drop",
    "Resource": "acs:odps:*:projects/prj1/tables/*"
  }
]
```

 **Note** The authorized user (alice@aliyun.com) can only submit a request from subnet 10.32.180.0/23 before 2017-11-11T23:59:59Z. The user can only perform the CreateInstance, CreateTable, and List operations on the prj1 project. The user cannot delete tables from prj1.

20.8.5. Differences between policy authorization and ACL authorization

ACL authorization:

- Use ACL authorization to grant or revoke permissions when both the grantee (such as a user or role) and the object (such as a table) exist. Like the security feature of Oracle authorization, this avoids the security risk out of dropping and recreating an object with the same name.
- When dropping an object, all authorizations related to the object are automatically revoked.
- It only supports allow (whitelist) authorization, and does not support deny (blacklist) authorization.
- Use the classic Grant/Revoke commands for authorization. The command is simple and not prone to mistakes. Conditional authorization is not supported.
- This method is suitable for simple scenarios where condition or deny is not needed for authorization, and only the existing objects need to be authorized.

Policy authorization:

- Use policy authorization to grant or revoke permissions when the grantee or object is not available. The Object parameter supports wildcard ". For example, *projects/tbproj/tables/taobao* matches all tables whose names start with taobao in project tbproj. Like the features of MySQL authorization, policy authorization allows a non-existent object to be authorized, so the authorizer must consider the security risks of dropping and recreating an object with the same name.
- When dropping an object, the policy authorization related to this object is not deleted.
- Both allow (whitelist) authorization and deny (blacklist) authorization are supported. If allow and deny conflict, the deny action takes priority.
- Conditional authorization is supported. The authorizer can enforce 20 conditions on allow or deny authorization. For example, these conditions can be used to limit access to IP addresses within a subnet, and allow access before 23:59:59 on November 11, 2017.
- This method is suitable for relatively complicated scenarios where conditional authorization and deny action are needed and non-existent objects need to be authorized.
- Using policy authorization is more complicated than ACL authorization, but provides more flexibility.

20.8.6. Application limits

Application limits

Item	Limit	Description
ACCESS_POLICY_SIZE_LIMIT	32 KB	The maximum size of AccessPolicy.
USER_NUMBER_LIMIT_IN_ONE_PROJECT	1,000	The maximum number of users in a project.

Item	Limit	Description
ROLE_NUMBER_LIMIT_IN_ONE_PROJECT	500	The maximum number of roles in a project.
ROLE_NAME_LENGTH_LIMIT	64	The maximum number of characters in a role name.
SECURITY_COMMENT_SIZE_LIMIT	1 KB	The maximum size of a security comment.
PACKAGE_NAME_LENGTH_LIMIT	128	The maximum number of characters in a package name.
ALLOW_PROJECT_NUMBER_LIMIT_IN_ONE_PACKAGE	1024	The maximum number of projects installed in a package.
RESOURCE_NUMBER_LIMIT_IN_ONE_PACKAGE	256	The maximum number of resources in a package.
PACKAGE_NUMBER_LIMIT_IN_ONE_PROJECT	512	The maximum number of packages that can be created in a project.
INSTALLED_PACKAGE_NUMBER_LIMIT_IN_ONE_PROJECT	64	The maximum number of packages that can be installed in a project.

20.9. Collection of security statements

20.9.1. Project security configuration

Authentication

Authorization configuration statements

Statement	Description
<code>show SecurityConfiguration</code>	Displays the project security configuration.
<code>set CheckPermissionUsingACL=true/false</code>	Enables or disables ACL-based authorization.
<code>set CheckPermissionUsingPolicy=true/false</code>	Enables or disables policy-based authorization.
<code>set ObjectCreatorHasAccessPermission=true/false</code>	Allows or disallows an object creator to be granted the object access permission by default.
<code>set ObjectCreatorHasGrantPermission=true/false</code>	Allows or disallows an object creator to be granted the ACL-based authorization permission by default.

Data protection

Data protection statements

Statement	Description
set ProjectProtection=false	Disables project protection.
set ProjectProtection=true [with exception <policy>]	Enables project protection.
list TrustedProjects	Displays the list of trusted projects.
add TrustedProject <projectName>	Adds a project to trusted projects.
remove trustedproject <projectname>;	Removes a project from trusted projects.

20.9.2. Project permission management

User management

User management statements

Statement	Description
list users	Lists all added users.
add user <username>	Adds a user.
remove user <username>	Removes a user.

Role management

Role management statements

Statement	Description
list roles	Lists all created roles.
create role <rolename>	Creates a role.
drop role <rolename>	Deletes a role.
grant <rolelist> to <username>	Revokes roles from a user.
revoke <rolelist> from <username>	Grants one or more roles to a user.

ACL-based authorization

ACL-based authorization statements

Statement	Description
<code>grant <privList> on <objType> <objName> to user <username></code>	Authorizes a user.
<code>grant <privList> on <objType> <objName> to role <rolename></code>	Authorizes a role.
<code>revoke <privList> on <objType> <objName> from user <username></code>	Revokes permissions from a user.
<code>revoke <privList> on <objType> <objName> from role <rolename></code>	Revokes permissions from a role.

Policy-based authorization

Policy-based authorization statements

Statement	Description
<code>get policy</code>	Displays policy settings at the project level.
<code>put policy <policyFile></code>	Configures a policy at the project level.
<code>get policy on role <roleName></code>	Displays the policy settings of a role.
<code>put policy <policyFile> on role <roleName></code>	Configures a policy for a role.

Permission review

Permission review statements

Statement	Description
<code>whoami</code>	Displays information about the current user.
<code>show grants [for <username>] [on type <objectType>]</code>	Displays permissions and roles of a user.
<code>show acl for <objectName> [on type <objectType>]</code>	Displays the authorization information of an object.
<code>describe role <roleName></code>	Displays the authorization and assignment information of a role.

20.9.3. Package-based resource sharing

Resource sharing

Resource sharing statements

Statement	Description
create package <pkgName>	Creates a package.
delete package <pkgName>	Deletes a package.
add <objType> <objName> to package<pkgName> [with privileges privs]	Adds resources that need to be shared to a package.
remove <objType> <objName> from package <pkgName>	Removes shared resources from a package.
allow project <prjName> to install package <pkgName> [using label <num>]	Allows a project to use a user package.
disallow project <prjName> to install package <pkgName>	Disables a project from using a user package.

Resource use

Resource use statements

Statement	Description
install package <pkgName>	Installs a package.
uninstall package <pkgName>	Uninstalls a package.

Package viewing

Package viewing statements

Statement	Description
show packages	Lists all created and installed packages.
describe package <pkgName>	Views details of a package.

21. Frequently-used tools

21.1. MaxCompute console

21.1.1. Usage notes

This topic provides usage notes for the MaxCompute client.

Notice

- Do not rely on the output data of the client in any development or planning processes, as the data format may change. The client output format may not be forward compatible. The command syntax and behavior vary according to versions.
- The MaxCompute client is a Java program. It requires JRE to run. You need to download and install JRE 1.8 to use the MaxCompute client.

For more information about how to configure and use the client, see [Quick start](#).

21.1.2. Install the client

This topic describes how to install the MaxCompute client.

1. Download the client package to your client computer.
2. Decompress the client package to a folder, where you can see the following folders:

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

3. Edit the following parameters in the `odps_config.ini` file in the `conf` folder:

```
project_name=  
access_id=*****  
access_key=*****  
end_point= <MaxCompute service address>
```

Note

- Set `access_id` and `access_key` to the AccessKey ID and AccessKey Secret of your cloud account.
- If you frequently use a project, enter the project name after `project_name=`. Then, you do not need to run the `use project_name;` command each time you log on to the client.

4. After the modifications, run the `odps` file in the `bin` directory (`./bin/odpscmd` in a Linux system or `./bin/odpscmd.bat` in a Windows system). Then, you are ready to run SQL statements. An example is as follows:


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

21.1.3. Configuration description

This topic describes some configurations and corresponding parameters of the MaxCompute client.

Help

Run the following command to view the help information about the client:

```
odps@ > ./bin/odpscmd -h;
```

 **Note** You can also enter `h;` or `help;` (case insensitive) in the interaction mode.

Startup parameters

Run the following command to specify a number of 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: (-f is used as an example)

1. Prepare a local script file named `script.txt`. The file is stored in `D:/`. Its contents are as follows:

```
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. The command output is as follows:

```
ID = 20170528122432906gux77io3
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-inc.com/api&podps_public_dev&i2017052 8122432906gux77io3&tokenRnlrSzJoL242YW43dFFIcldmb1ZWZzFxFxQlRFPsXPRFBTX09CTzoxMDcwMDI1NjI3ODA1 NjI5LDE0MzM0MjA2NzMseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWNOIjoiQWxs3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB yb2plY3RzL29kcHNfcHVibGljX2Rld i9pbN0Yw5jZXMvMjAxNTA1MjgxMjI0MzI5MDZndXg3N2lvMyJdfV0sIlZlcnNpb24iOiIxIn0=
OK
ID = 20170528122439318gcmkk6ul
Log view:
http://webconsole.odps.aliyun-inc.com:8080/logview/?h=http://service-corp.odps.aliyun-inc.com/api&podps_public_dev&i2017052 8122439318gcmkk6ul&tokenSt0RXdlV0M5YjZET2I1MnJuU FkzWDN1aWpzPSxPRFBTX09CTzoxMDcwMDI1NjI3ODA1NjI5LDE0MzM0MjA2ODaseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWNOIjoiQWxs3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB yb2plY3RzL29kcHNfcHVibGljX2Rld i9pbN0Yw5jZXMvMjAxNTA1MjgxMjI0MzI5MDZndXg3N2lvMyJdfV0sIlZlcnNpb24iOiIxIn0=
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&i2017052 8122440389g98cmlmf&tokenNWlwL0EvQThxUXhzcTRERDc5N Fg0b2IxZ3QwPSxPRFBTX09CTzoxMDcwMDI1NjI3OD A1NjI5LDE0MzM0MjA2ODaseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWNOIjoiQWxs3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnB yb2plY3RzL29kcHNfcHVibGljX2Rld i9pbN0Yw5jZXMvMjAxNTA1MjgxMjI0NDZlbn0ThjbWxtZiJdfV0sIlZlcnNpb24iOiIxIn0=
OK
```

Interactive mode

Directly run the client, and you will enter the interaction mode. The following information is displayed:

```
[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 at the cursor (terminated by a semicolon), and press Enter to run it.

Command output

The output of a SQL statement is in either HumanReadable (default) or MachineReadable format. If you use the -M parameter when running odpscmd, the output format is CSV.

 **Note** This function currently applies only to SELECT and READ statements, and takes effect when reading data.

Continue run

When you run `odpscmd` in `-e` or `-f` mode and want to start with an intermediate statement among a few statements, you can use the `-k` parameter. This parameter indicates that the execution starts from the specified statement while the preceding statements are skipped. If the parameter value is equal to or smaller than 0, execution starts from the first statement. A statement terminated by a semi-colon is considered a valid statement. At runtime, the process indicates the specific statement that is running successfully or fails.

For example, the `/tmp/dual.sql` file includes the following three 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 ignore the first two statements:

```
odpscmd -k 3 -f dual.sql
```

Obtain information about the current logon user

Run the following command to obtain the cloud account of the current logon user and the endpoint that is used:

```
whoami
```

Example:

```
odps@ hiveut>whoami; Name: odpstest@aliyun.com ID: 1090142773636588 End_Point: <MaxCompute
service address> Project: lijunsecuritytest
```

Exit

Command syntax:

```
odps@ > quit;
```

or

```
q;
```

Configure a job priority

Command syntax:

```
Admin@ > ./bin/odpscmd --instance-priority=<PRIORITY>;
```

Configuration file: `odps_config.ini`

```
instance_priority=<PRIORITY>
```

 Notice

- The value range of <PRIORITY> is 0-9, where 0 indicates the highest priority and 9 the lowest priority.
- The priority setting in the configuration file applies to all instances submitted in CLT.
- The priority value is 9 by default.

DryRun mode

In the DryRun mode, MaxCompute parses a SQL statement to check if the syntax is correct and generates an execution plan. MaxCompute does not submit a distributed job. Command syntax:

```
./bin/odpscmd -y
```

SQL reliability

If an exception is returned during the execution of SQL statements such as INSERT or CREATE TABLE AS, the client tries to automatically recover data and metadata to the pre-execution state based on known information.

- Data that is overwritten by INSERT OVERWRITE during QUERY execution is recovered from the temporary backup directory to the original directory.
- Data that is generated by INSERT INTO during QUERY execution is removed.
- Tables created during QUERY execution and dynamically generated partition information are removed.

If MaxCompute fails to recover data, it returns a specific error code. The error code alerts you that further attempts can make some data unrecoverable. The error code is as follows:

```
ODPS-  
0110999: Critical! Internal error happened in commit operation and rollback failed, possible breach of atomicity
```

21.2. Eclipse development plugin

21.2.1. Install Eclipse

This topic describes how to install the Eclipse plug-in.

Context

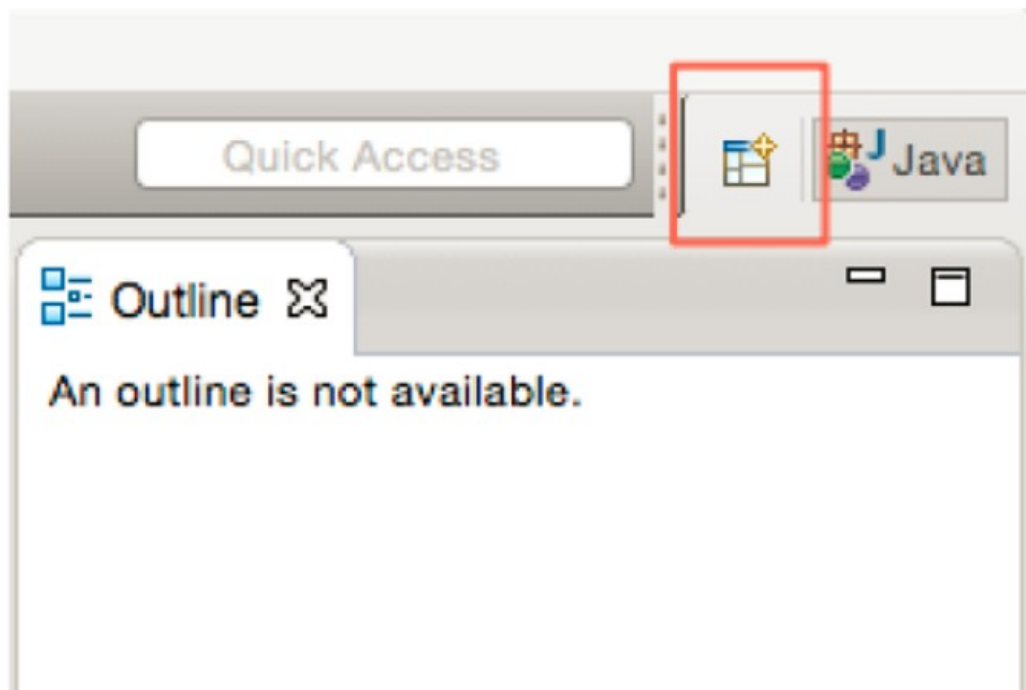
MaxCompute provides the Eclipse plug-in to help you easily use the Java SDKs for MapReduce and UDFs in your development work. This plug-in can simulate the running process of MapReduce or UDFs. It provides local debugging methods and simple template generation features. You can download the software package from [Eclipse](#).

 **Note** Unlike the local running mode of MapReduce, the Eclipse plug-in cannot synchronize data with MaxCompute. You need to manually copy the required data to the warehouse directory of Eclipse.

Procedure

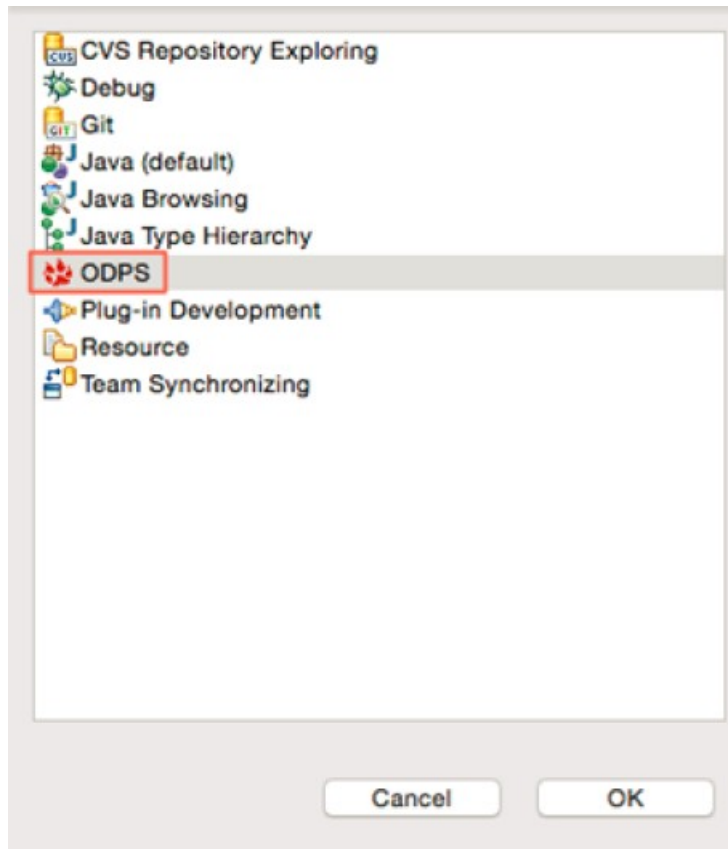
1. Decompress the Eclipse package to obtain the following JAR file:
odps-eclipse-plugin-bundle-0.15.0.jar
2. Place the JAR file in the plugins folder under the Eclipse installation directory.
3. Start Eclipse and click the Open Perspective icon in the upper-right corner, as shown in the following figure.

Eclipse installation 1

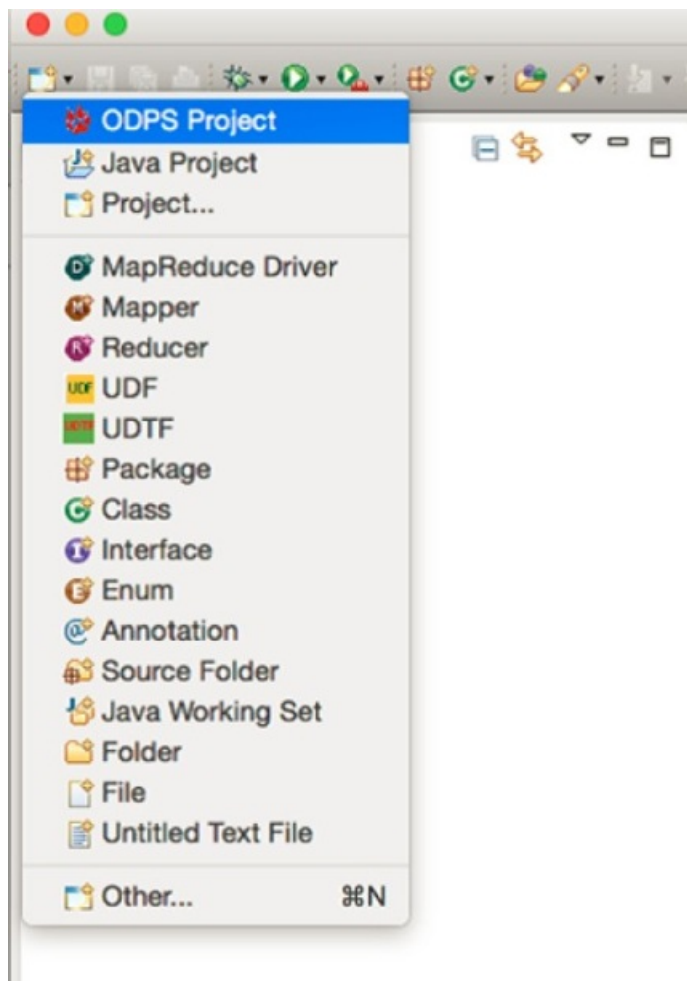


4. Click **ODPS** in the dialog box that appears, as shown in the following figure.

Eclipse installation 2

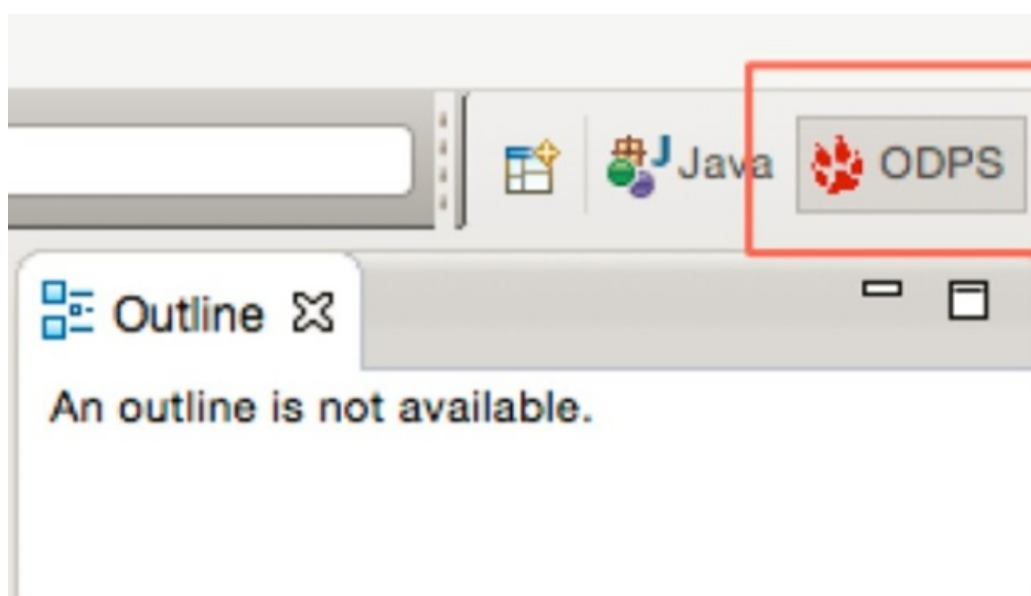


5. After you click **ODPS**, a navigation pane appears, as shown in the following figure.
Eclipse installation 3



6. Click **ODPS Project** in the navigation pane and then 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.

Eclipse installation 4



21.2.2. Create a project

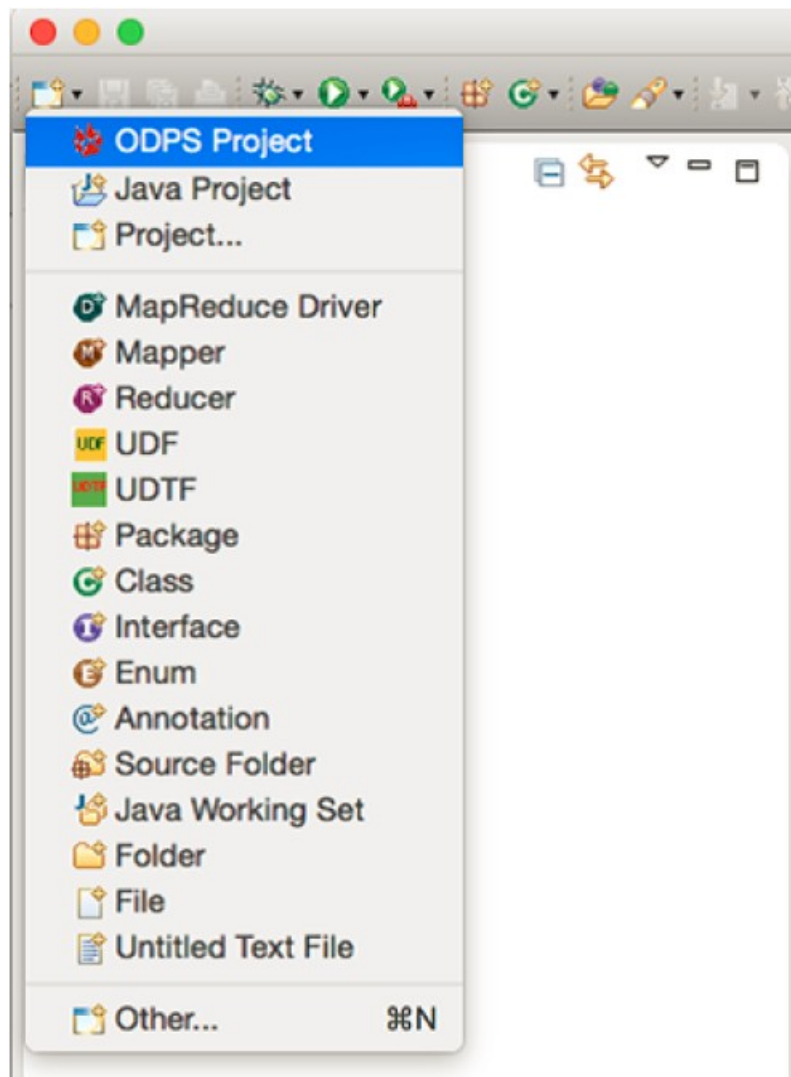
21.2.2.1. Method 1

This topic describes the first method to create a project in the Eclipse development plugin.

Procedure

1. Start Eclipse. Choose **File > New > Project > ODPS > ODPS Project** in the upper-left corner to create a project, as shown in the following figure.

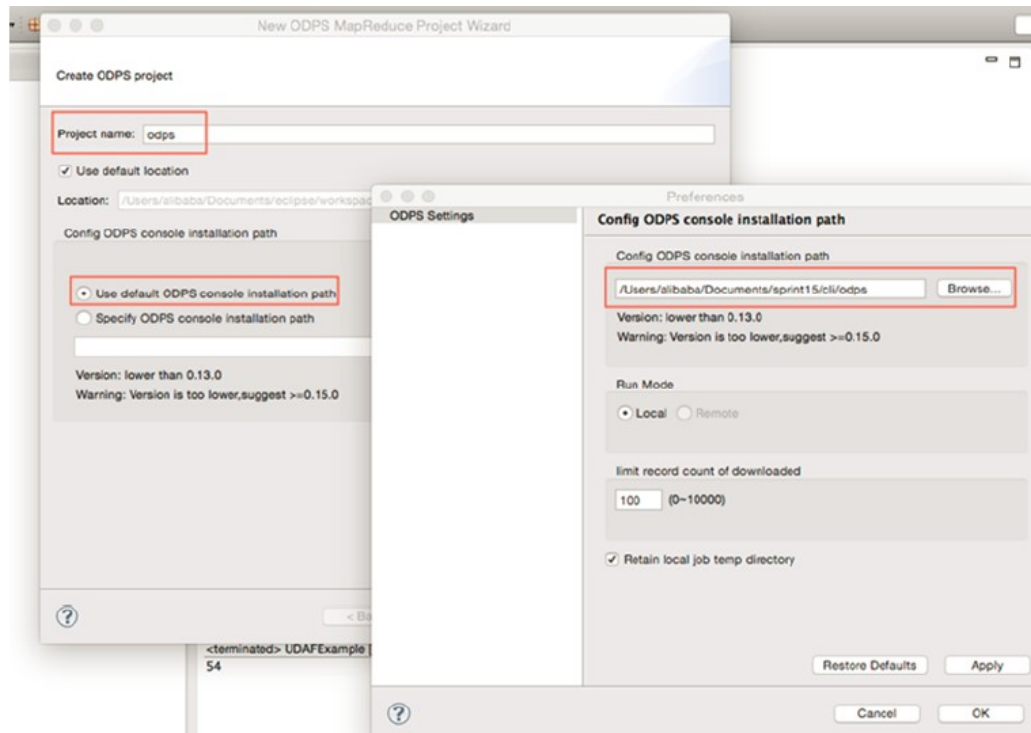
Step 1



 **Note** In this example, the project name is ODPS.

2. After the ODPS project is created, a dialog box is displayed, as shown in the following figure. Set Project name, select the path of the MaxCompute client, and then click **Finish**.

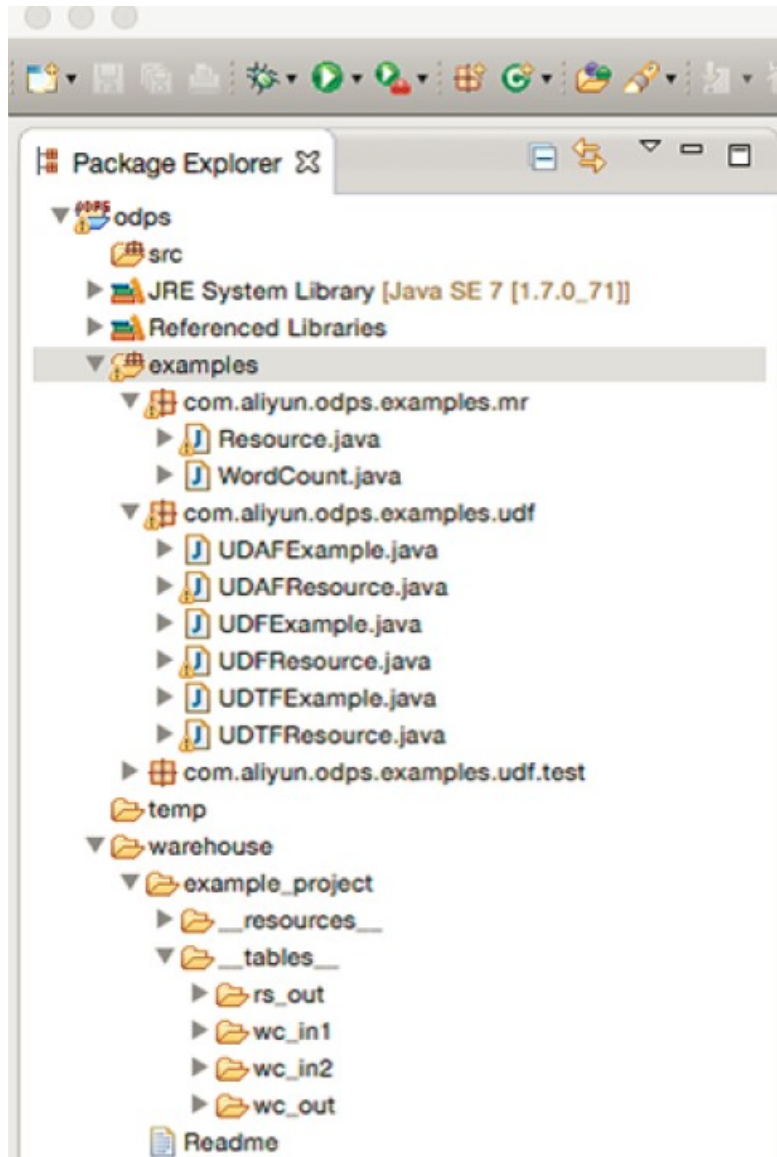
Step 2



 **Note** The client must be installed in advance.

3. After creating a project, you can see the directory structure on the left-side Package Explorer pane, as shown in the following figure.

Step 3



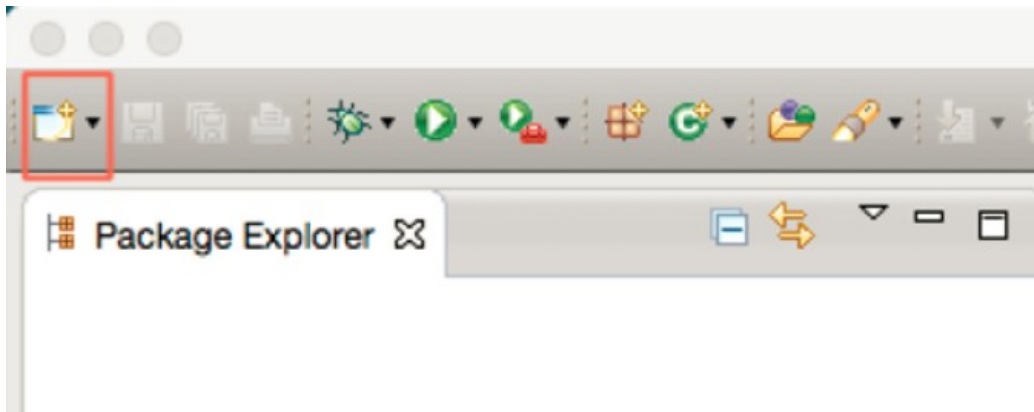
21.2.2.2. Method 2

This topic describes the second method to create a project in the Eclipse development plugin.

Procedure

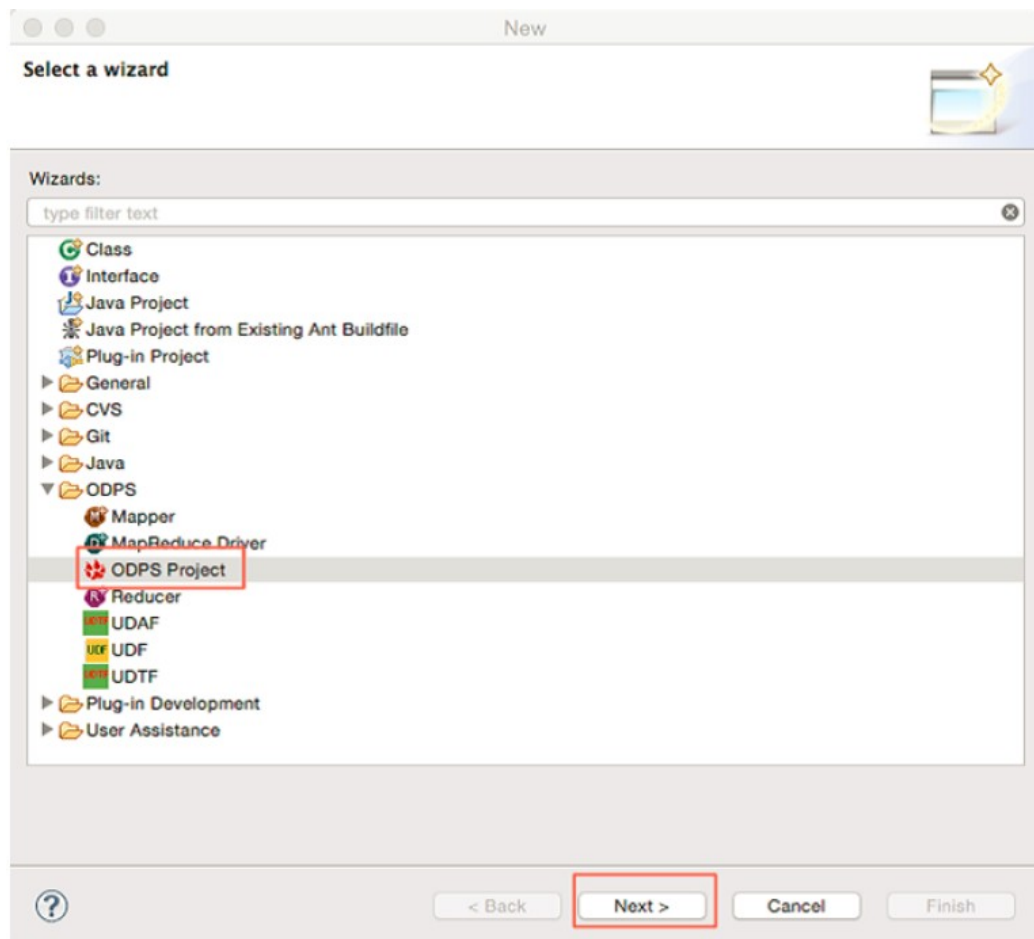
1. Start Eclipse. Click **New** in the upper-left corner, as shown in the following figure.

Step 1



2. In the dialog box that appears, select **ODPS Project** and click **Next**, as shown in the following figure.

Step 2



 **Note** In this example, the project name is ODPS.

3. The subsequent steps are the same as those in method 1. After installing the Eclipse plugin, you can use it to compile MapReduce or UDF programs.

 **Note** For a MapReduce running example, see [MapReduce running example](#). For a UDF development and running example, see [UDF development and running example](#).

21.2.3. MapReduce running example

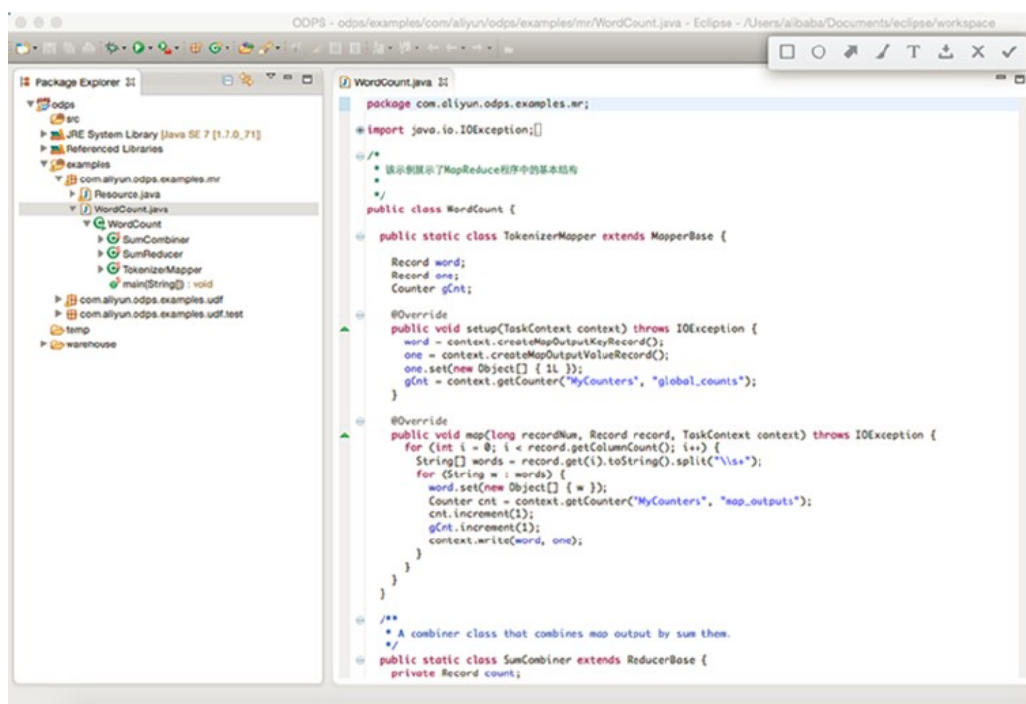
21.2.3.1. Quickly run a WordCount example

This topic describes how to use MapReduce to quickly run a WordCount example in the Eclipse plugin.

Procedure

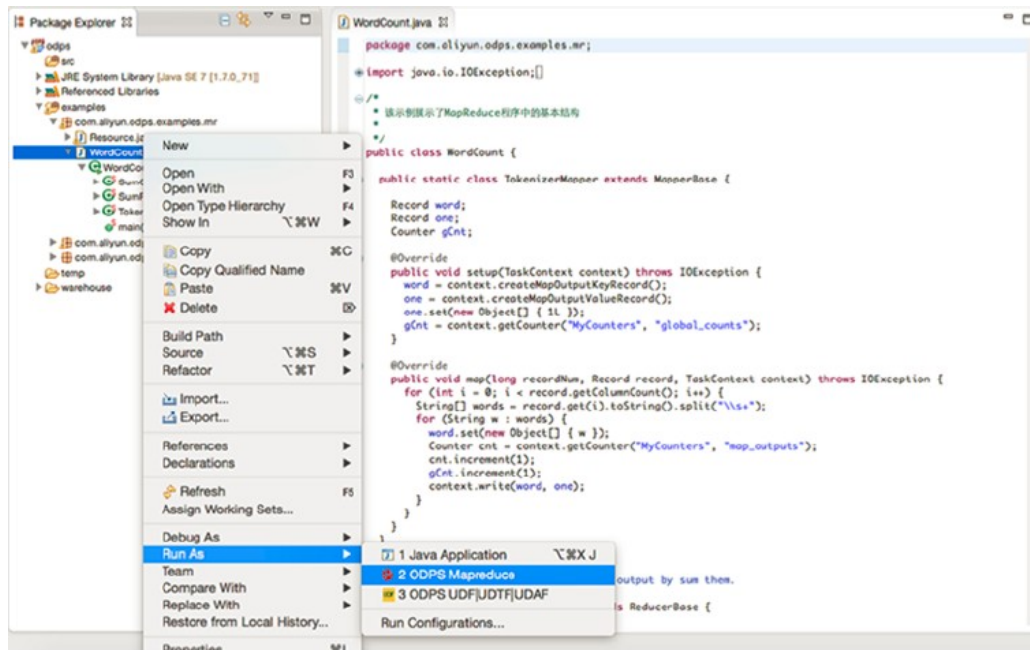
1. Select a WordCount example in MaxCompute, as shown in the following figure.

WordCount example



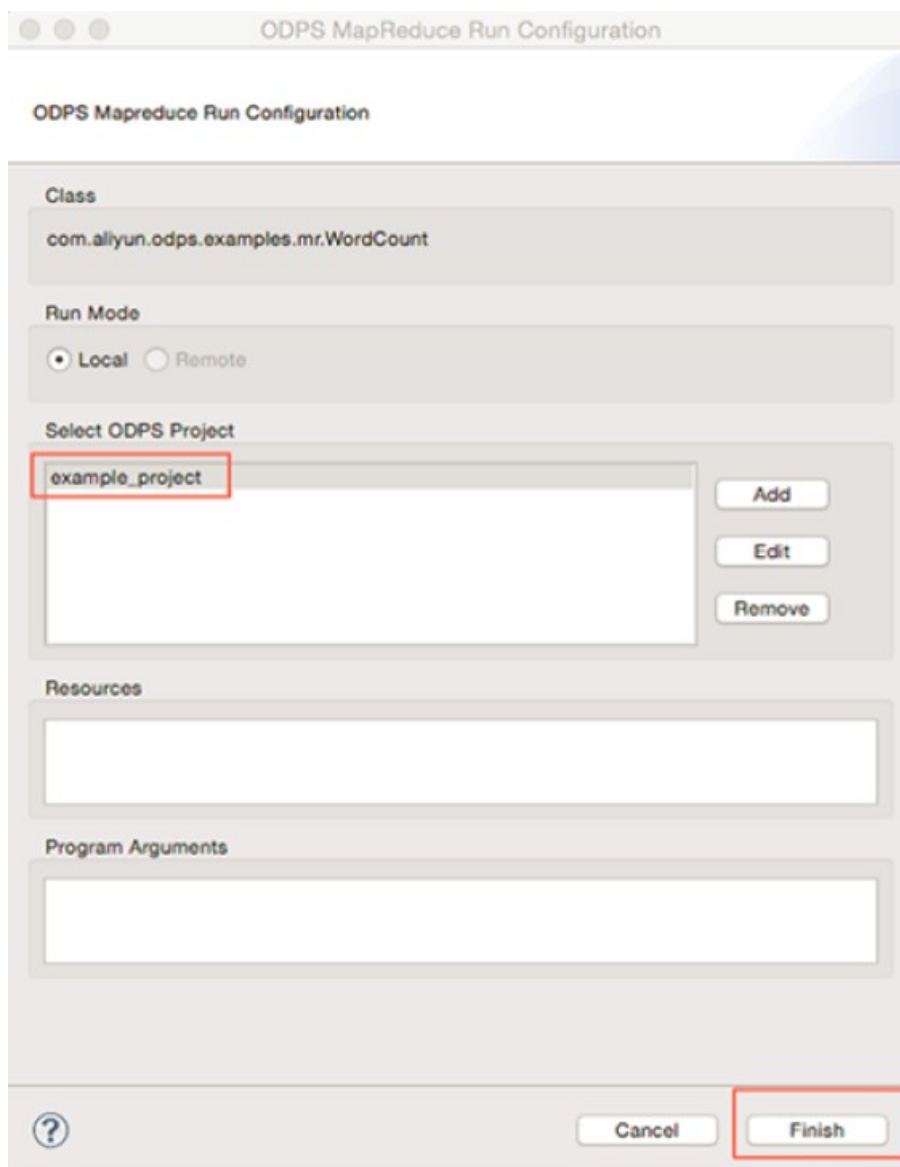
2. Right-click **WordCount.java** and choose **Run AsODPS MapReduce** from the shortcut menu, as shown in the following figure.

Run the WordCount example



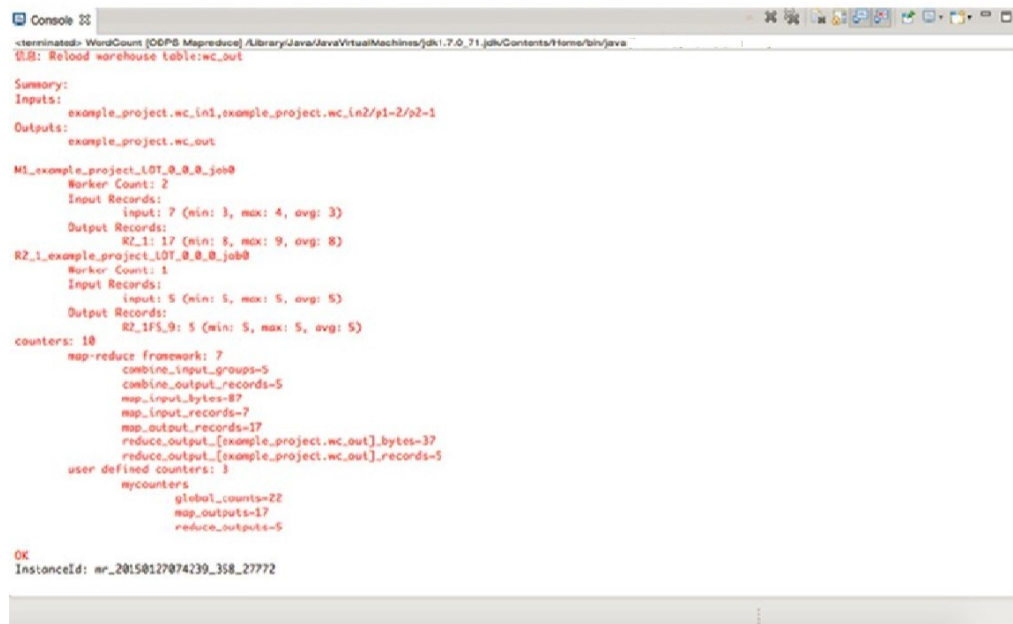
3. In the dialog box that appears, select **example_project** and click **OK**, as shown in the following figure.

Run the WordCount example



4. The results of the operation are displayed after the operation is executed, as shown in the following figure.

Execution result of the WordCount example



```

<terminated> WordCount [ODPS MapReduce] /Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/bin/java
00:08: Reload warehouse table:wc_out

Summary:
Inputs:
    example_project.wc.in1,example_project.wc.in2/p1-2/p2-1
Outputs:
    example_project.wc_out

M1_example_project_LOT_0_0_0_job0
  Worker Count: 2
  Input Records:
    input: 7 (min: 1, max: 4, avg: 3)
  Output Records:
    RZ_L1: 17 (min: 8, max: 9, avg: 8)
R2_1_example_project_LOT_0_0_0_job0
  Worker Count: 1
  Input Records:
    input: 5 (min: 5, max: 5, avg: 5)
  Output Records:
    RZ_IFS_0: 5 (min: 5, max: 5, avg: 5)

counters: 10
  map-reduce framework: 7
    combine_input_groups=5
    combine_output_records=5
    map_input_bytes=87
    map_input_records=7
    map_output_records=17
    reduce_output_[example_project.wc_out].bytes=37
    reduce_output_[example_project.wc_out].records=5
  user defined counters: 3
    mycounters
      global_counts=22
      map_outputs=17
      reduce_outputs=5

OK
InstanceId: mr_20150127074239_358_27772

```

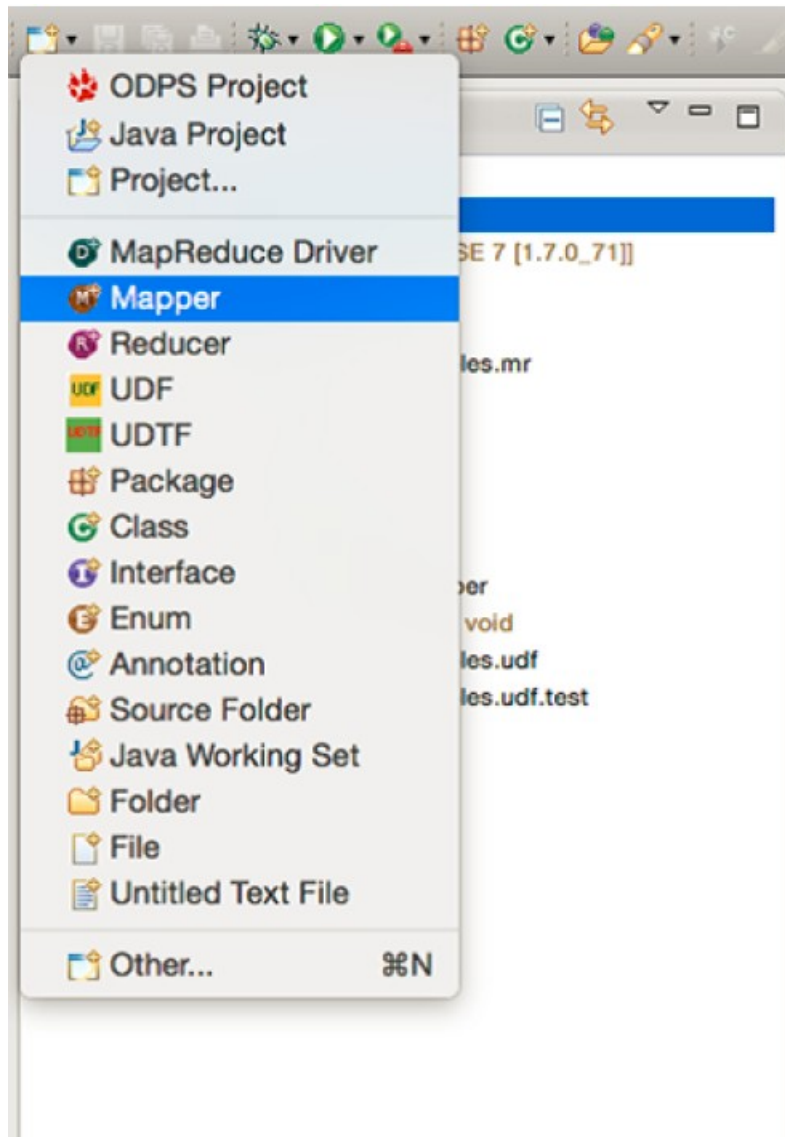
21.2.3.2. Run a custom MapReduce program

This topic provides an example on how to run a custom MapReduce program in the Eclipse plugin.

Procedure

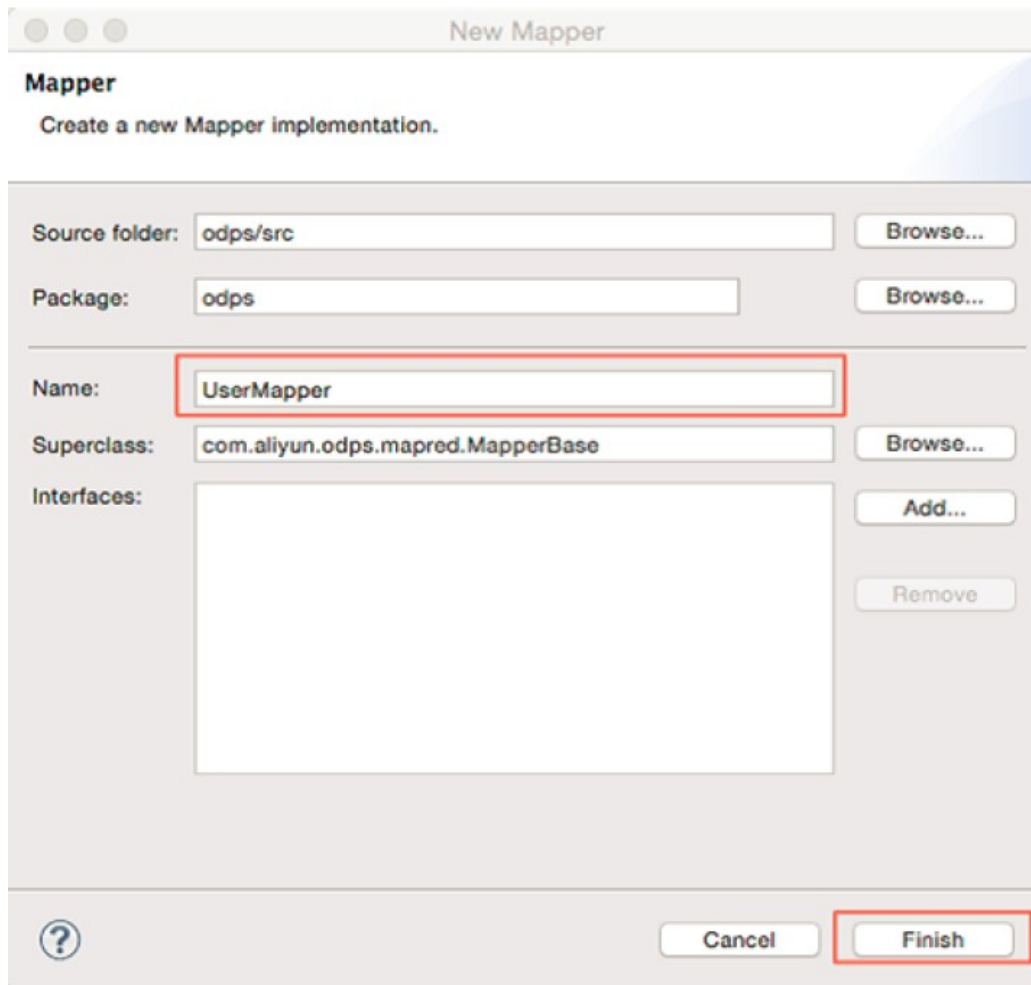
1. In Eclipse, right-click the `src` directory, and choose **New > Mapper** from the short cut menu, as shown in the following figure.

Step 1



2. Select Mapper. A dialog box is displayed, as shown in the following figure. Enter the name of the Mapper class, and click Finish.

Step 2

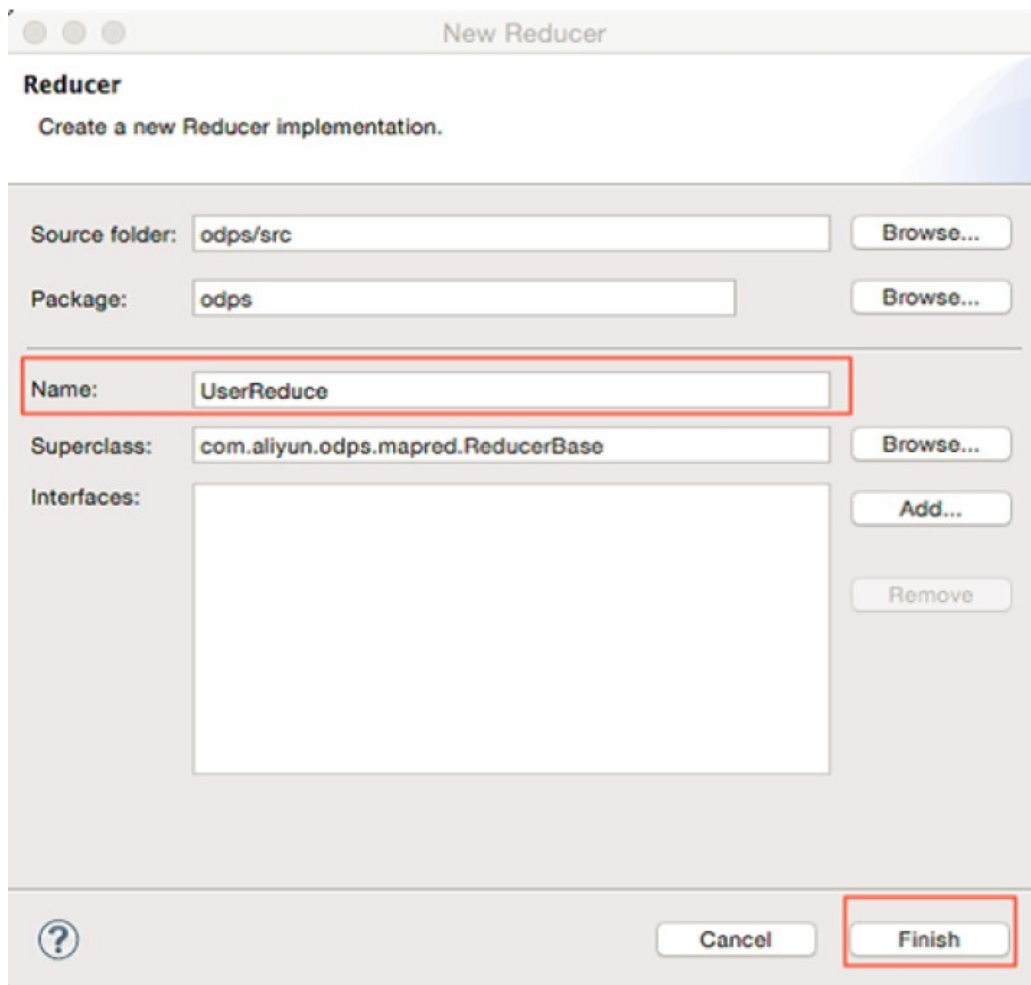


3. On the left-side Package Explorer, the UserReducer.java file is generated in the src directory. The file contains a Mapper class template. The package name is odps by default. The template content is as follows.

```
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. In Eclipse, right-click the **src** directory, and choose **New > Reduce** from the shortcut menu, as shown in the following figure.

Reducer



5. In the dialog box that appears, enter the name of the Reduce class and click Finish.

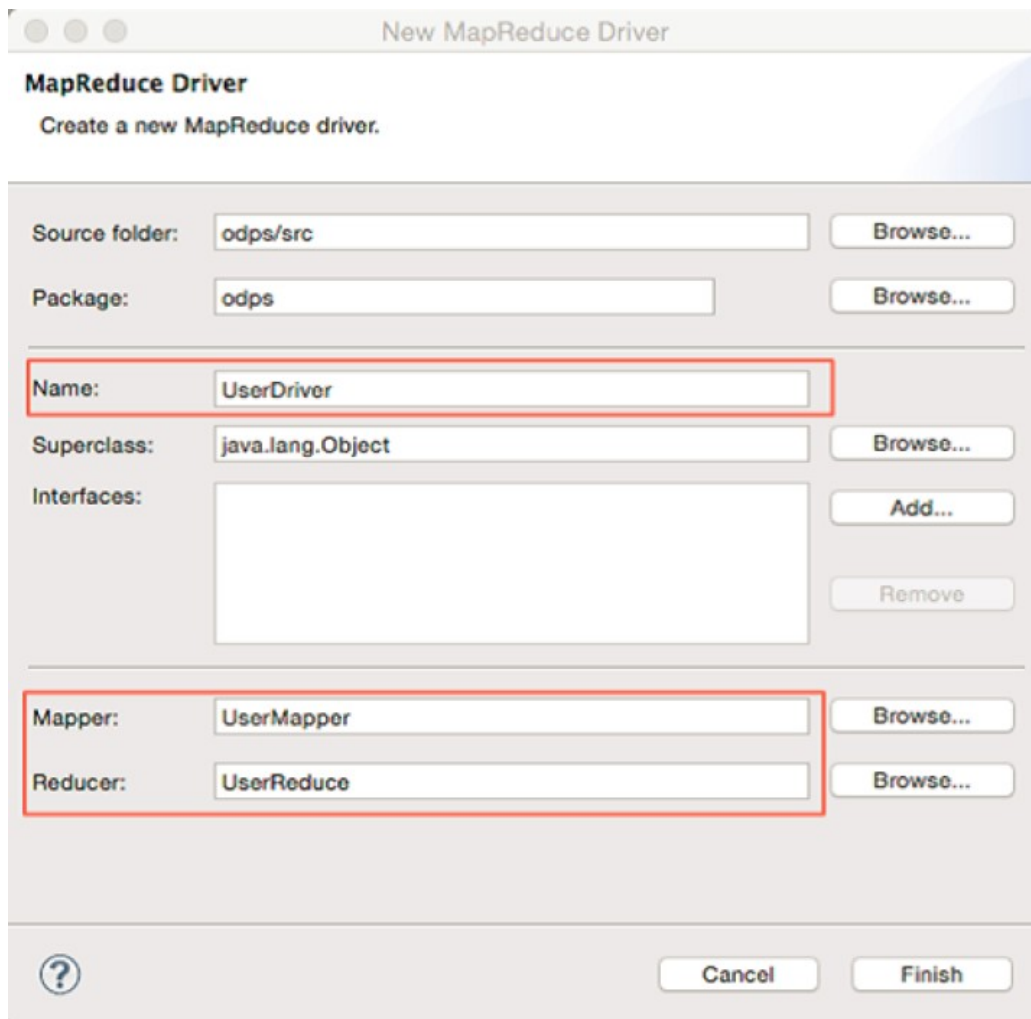
 **Note** This example uses UserReducer.

6. On the left-side Package Explorer, the UserReducer.java file is generated in the src directory. The file contains a Reduce class template. The package name is odps by default. The template content is as follows:

```
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. In Eclipse, right-click the **src** directory, and choose **New > MapReduce Driver** from the shortcut menu.
8. A dialog box is displayed, as shown in the following figure. Set Name, Mapper, and Reducer, and then click Finish.

MapReduce Driver

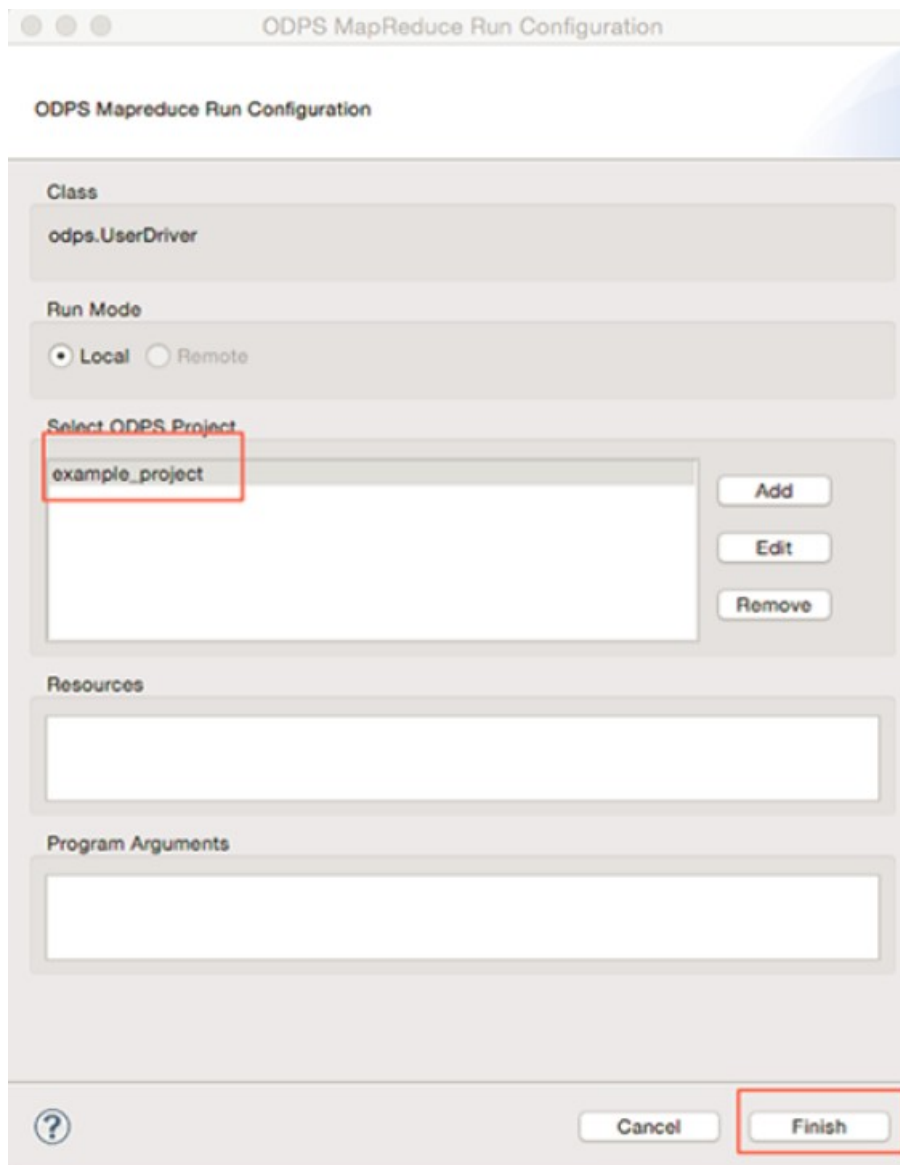


9. On the left-side Package Explorer, the MyDriver.java file is generated in the src directory. The file contains a MapReduce Driver template. The package name is odps by default. The template content is as follows:

```
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 UserDriver.java and choose **Run As > ODPS MapReduce** from the shortcut menu. In the dialog box that appears, click **OK**. A dialog box is displayed, as shown in the following figure.

ODPS MapReduce Run Configuration



11. Select `example_project` as the MaxCompute project. Click **Finish** to start running the MapReduce program locally. If the output is as shown in the following figure, the local running is successful.

Console

```

<terminated> UserDriver [ODPS MapReduce] /Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/bin/java

Summary:
Inputs:
    example_project.wc_in1,example_project.wc_in2/p1=2/p2=1
Outputs:
    example_project.wc_out

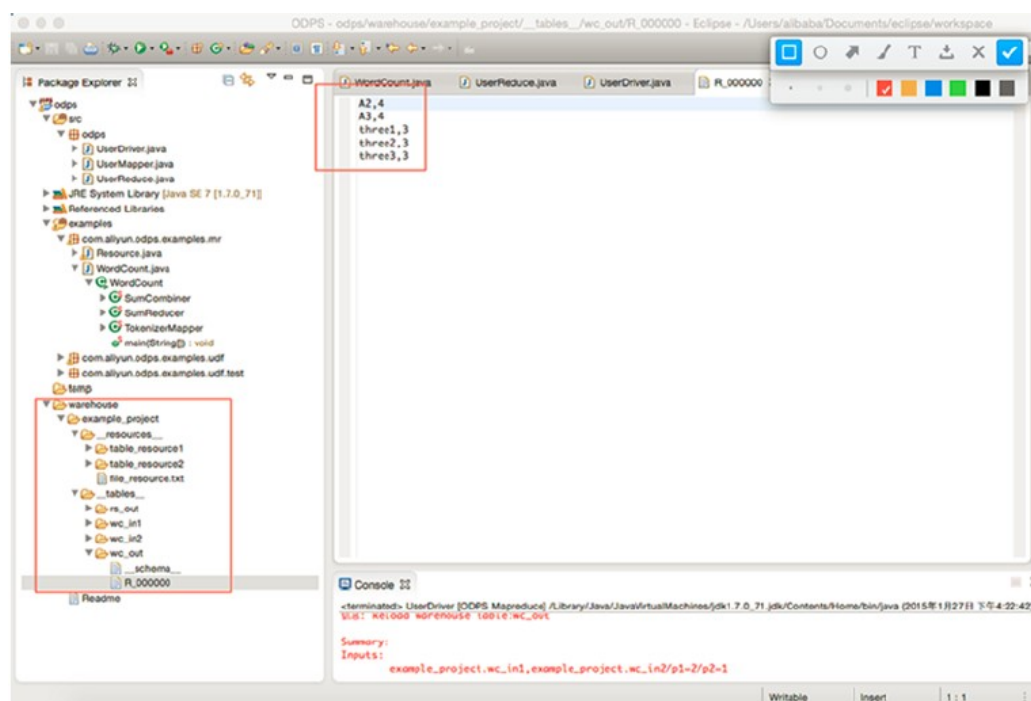
M1_example_project_L0T_0_0_0.job0
  Worker Count: 2
  Input Records:
    input: 7 (min: 3, max: 4, avg: 3)
  Output Records:
    R2_1: 17 (min: 8, max: 9, avg: 8)
R2_1_example_project_L0T_0_0_0.job0
  Worker Count: 1
  Input Records:
    input: 5 (min: 5, max: 5, avg: 5)
  Output Records:
    R2_1FS_9: 5 (min: 5, max: 5, avg: 5)
counters: 18
  map-reduce framework: 7
    combine_input_groups=5
    combine_output_records=5
    map_input_bytes=87
    map_input_records=7
    map_output_records=17
    reduce_output_[example_project.wc_out].bytes=37
    reduce_output_[example_project.wc_out].records=5
  user defined counters: 3
    mycounters
      global_counts=22
      map_outputs=17
      reduce_outputs=5

OK
InstanceId: wr_28158177882243_694_27864

```

12. The execution result is stored in the warehouse directory. Refresh the ODPS project, as shown in the following figure.

Output

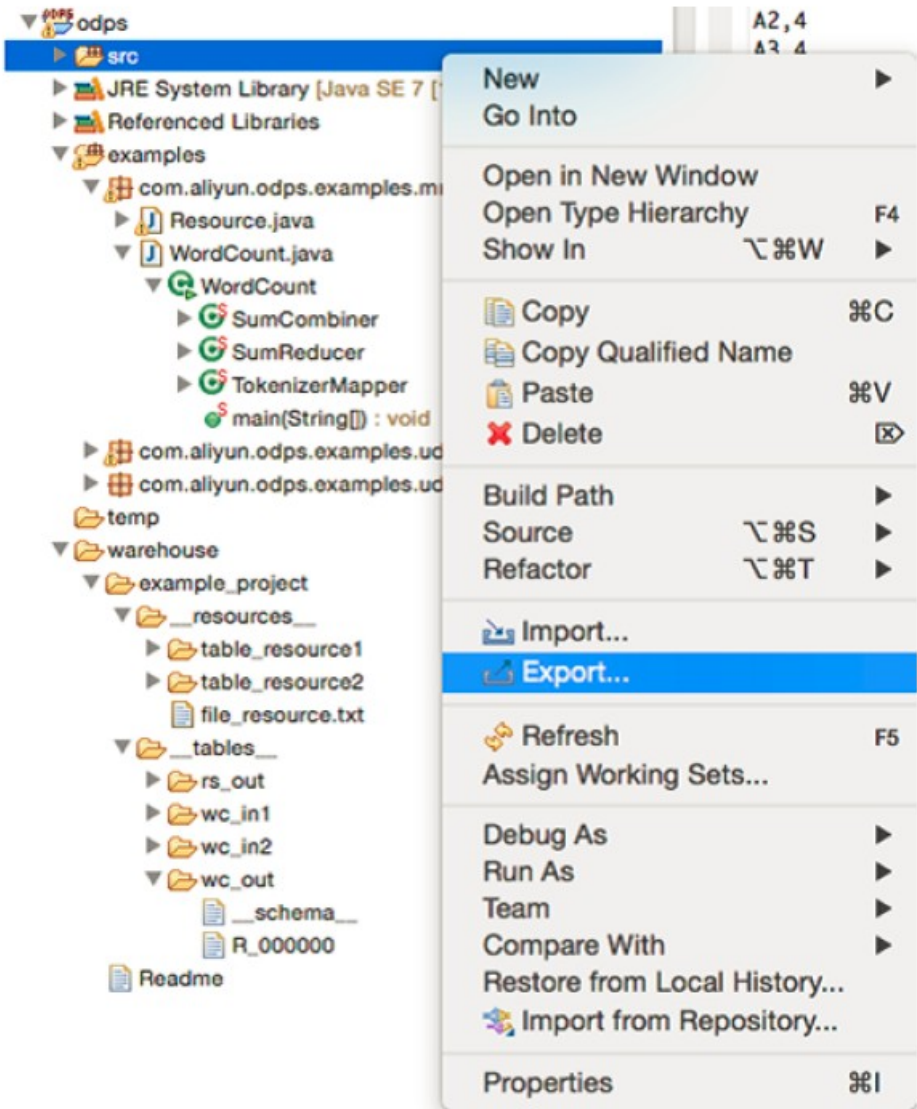


Note wc_out is the output directory, and R_000000 is the result file. After confirming that the result is correct through local debugging, you can use the export function of Eclipse to package the MapReduce program for subsequent use in the distributed environment.

13. The export procedure is as follows:

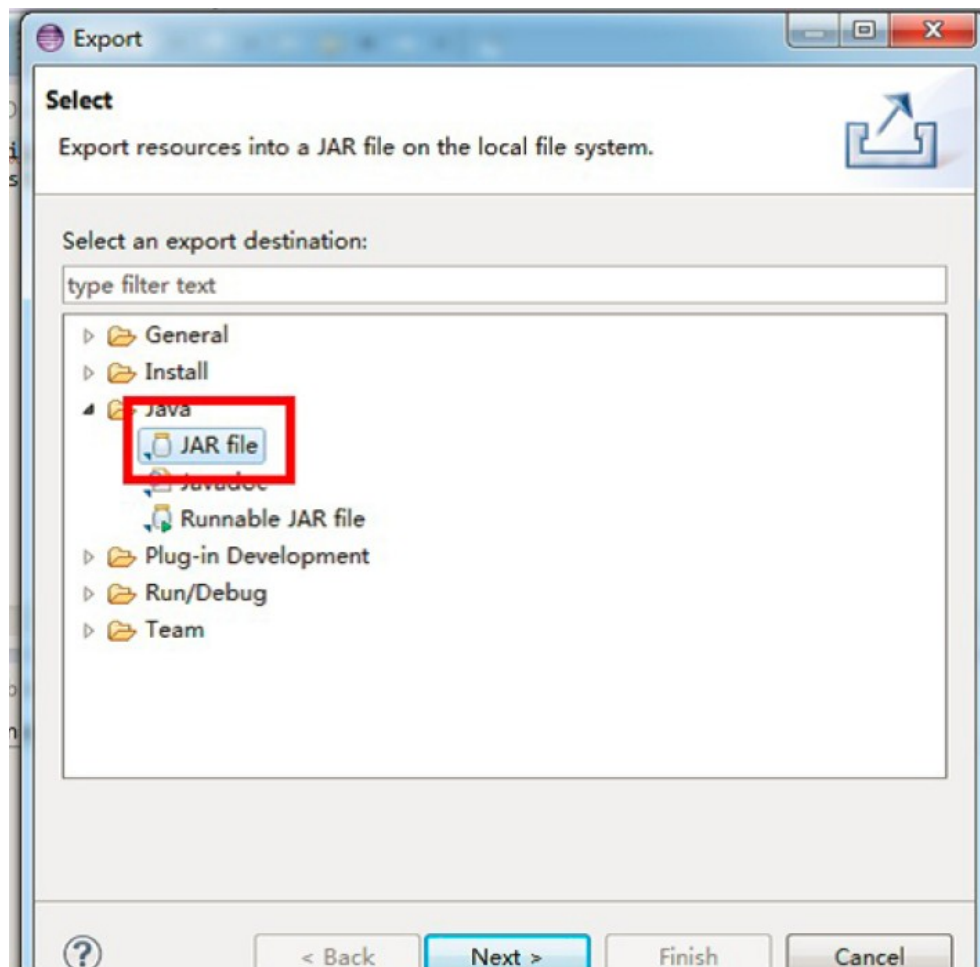
- i. Right-click the `src` directory and select **Export** from the short cut menu, as shown in the following figure.

Step 1



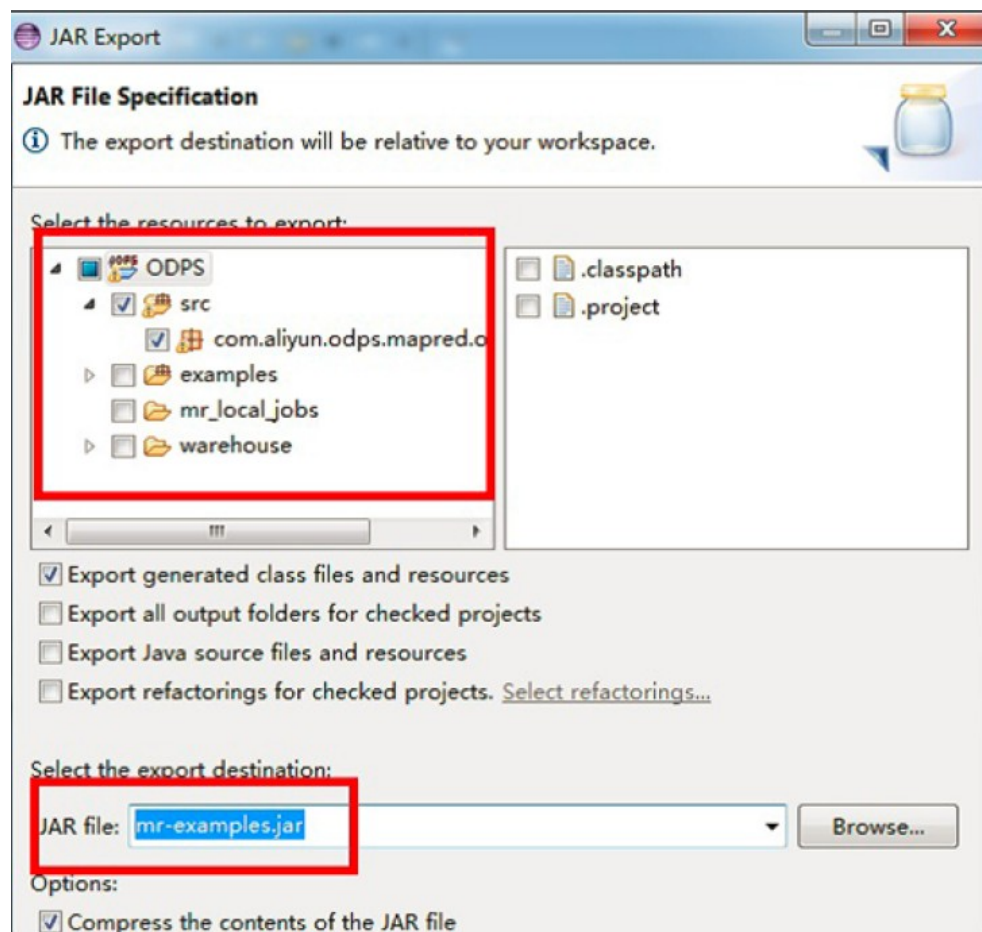
- ii. Select JAR file as the export destination, as shown in the following figure.

Step 2



- iii. You only need to export the package (com.aliyun.odps.mapred.open.example) in the src directory. Set JAR file to mr-examples.jar, as shown in the following figure.

Step 3



 **Note** In this example, the program package is named mr-examples.jar. You can name the package based on your actual requirements.

- iv. Click OK. The export process is complete.
14. If you want to create a new project locally, you can create a new subdirectory (at the same level as example_project) under warehouse. 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

```

An example of the schema file:

Non-partitioned table:

```

project=project_name table=table_name
columns=col1:BIGINT,col2:DOUBLE,col3:BOOLEAN,col4:DATETIME,col5:STRING
-- 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
-- Note that the following data formats are supported: bigint, double, boolean, datetim
e, and string. These formats correspond to the following Java data types: long, double,
boolean, java.util.Date, and java.lang.String.

```

An example of the data file:

```

1,1.1,true,2015-06-04 11:22:42 896,hello world
\n,\n,\n,\n,\n
-- Note that the time is in milliseconds. For all time formats, \N is used to represent
NULL.

```

 Note

- Run the MapReduce program locally. The warehouse directory is checked for data tables or resources by default. If the tables or resources do not exist, data of the server is downloaded to the warehouse directory. Then the program runs locally.
- After running MapReduce, refresh the warehouse directory to view the generated result.

21.2.4. UDF development and running example

21.2.4.1. Local debug UDF programs

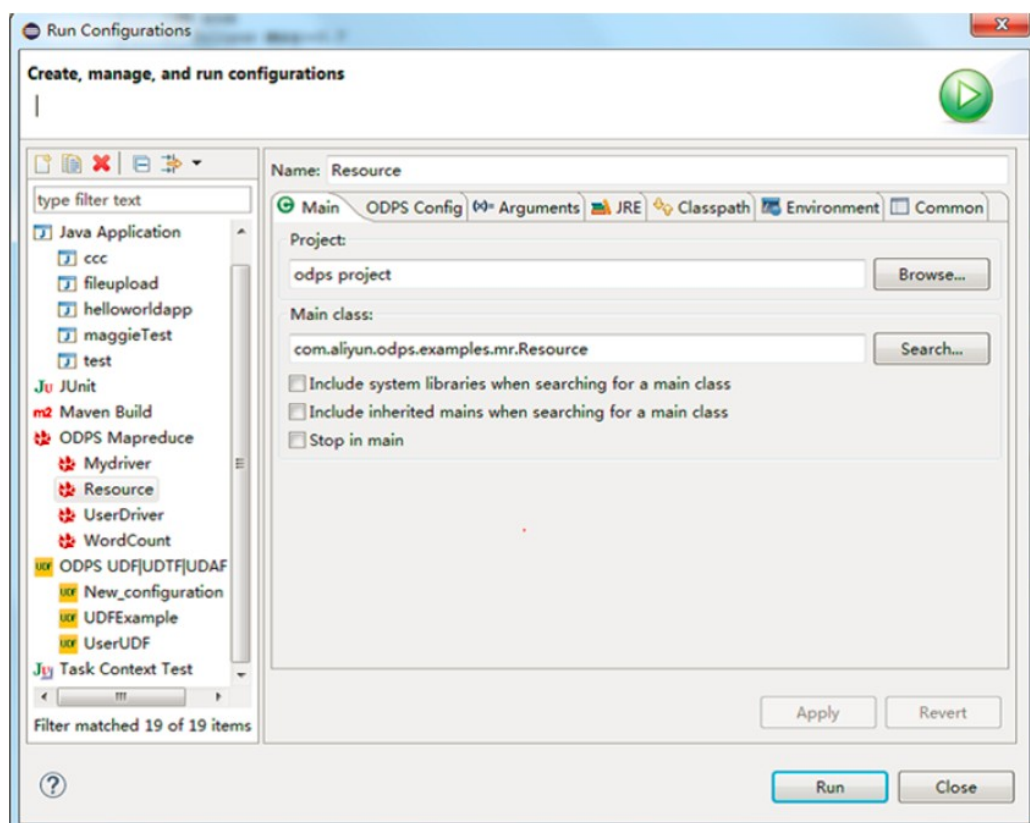
21.2.4.1.1. Run a UDF from the menu bar

This topic describes how to quickly run a UDF from the menu bar of the Eclipse plugin.

Procedure

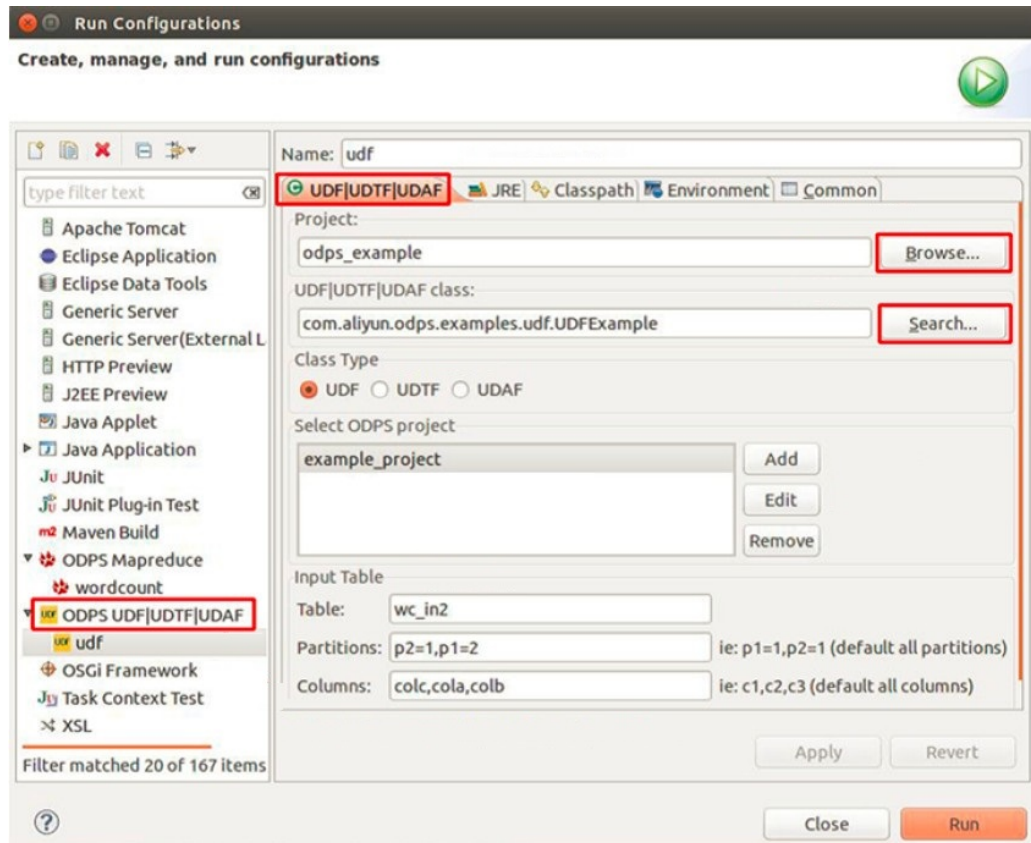
1. Choose **Run > Run Configurations** from the menu bar. A dialog box is displayed, as shown in the following figure.

Run Configurations 1



2. To create a run configuration, select the UDF class and type to be run, select an ODPS project, and fill in the input table information, as shown in the following figure.

Run Configurations 2



Note There are three parameters in the Input Table area: Enter the input table of the UDF in Table. Enter the partitions from which data is read in Partitions. Separate multiple partitions by commas (.). Enter the columns in which data is transmitted as UDF parameters in Columns. Separate multiple columns by commas (.).

3. Click **Run**. The running result is displayed in the console, as shown in the following figure.

Console

```
<terminated> udf [ODPS UDF|UDTF|UDAF] /home/shihai/lib/java/bin/java (Dec 12, 2014 7:32:23 PM)
sss2s: three3,three1,three2
sss2s: three3,three1,three2
sss2s: three3,three1,three2
```

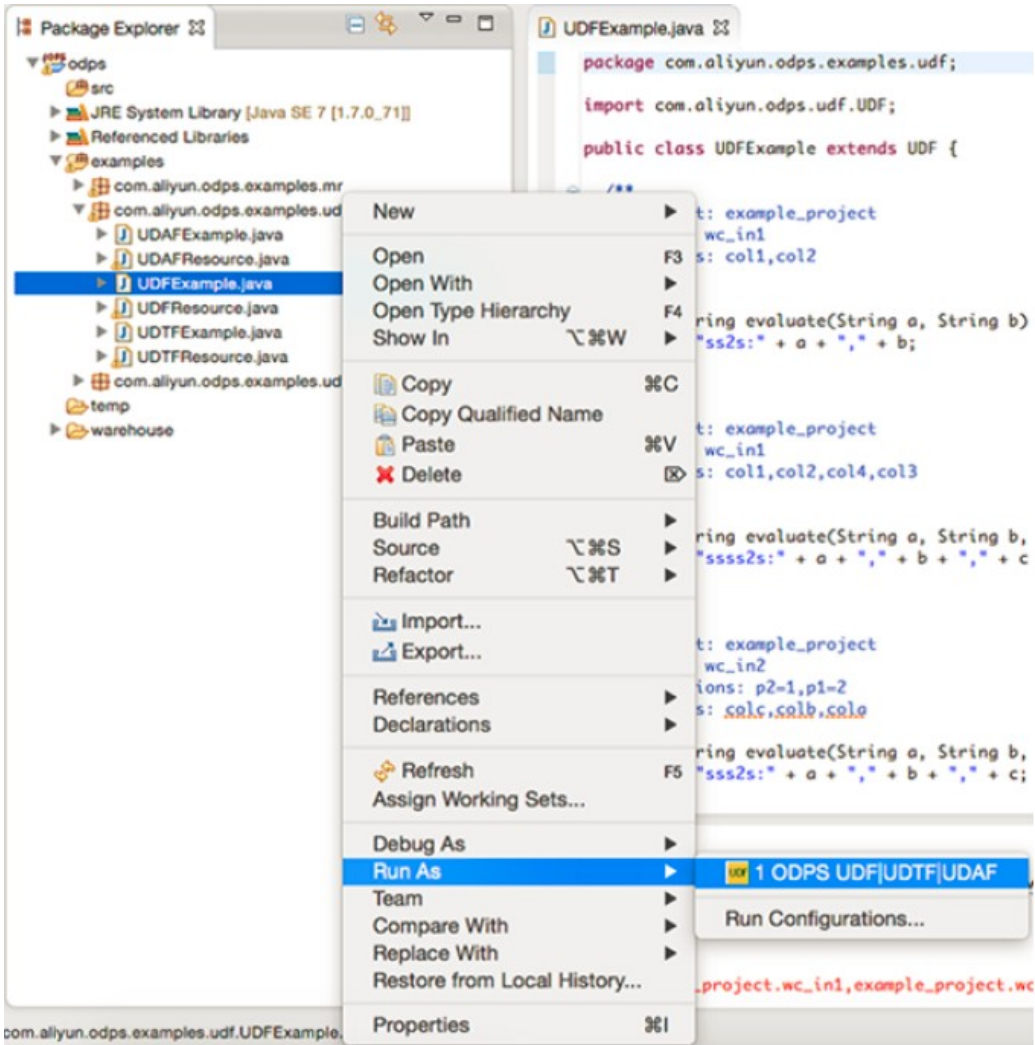
21.2.4.1.2. Use the right-click shortcut menu to quickly run a UDF

This topic describes how to use the right-click shortcut menu to quickly run a UDF in the Eclipse development plugin.

Procedure

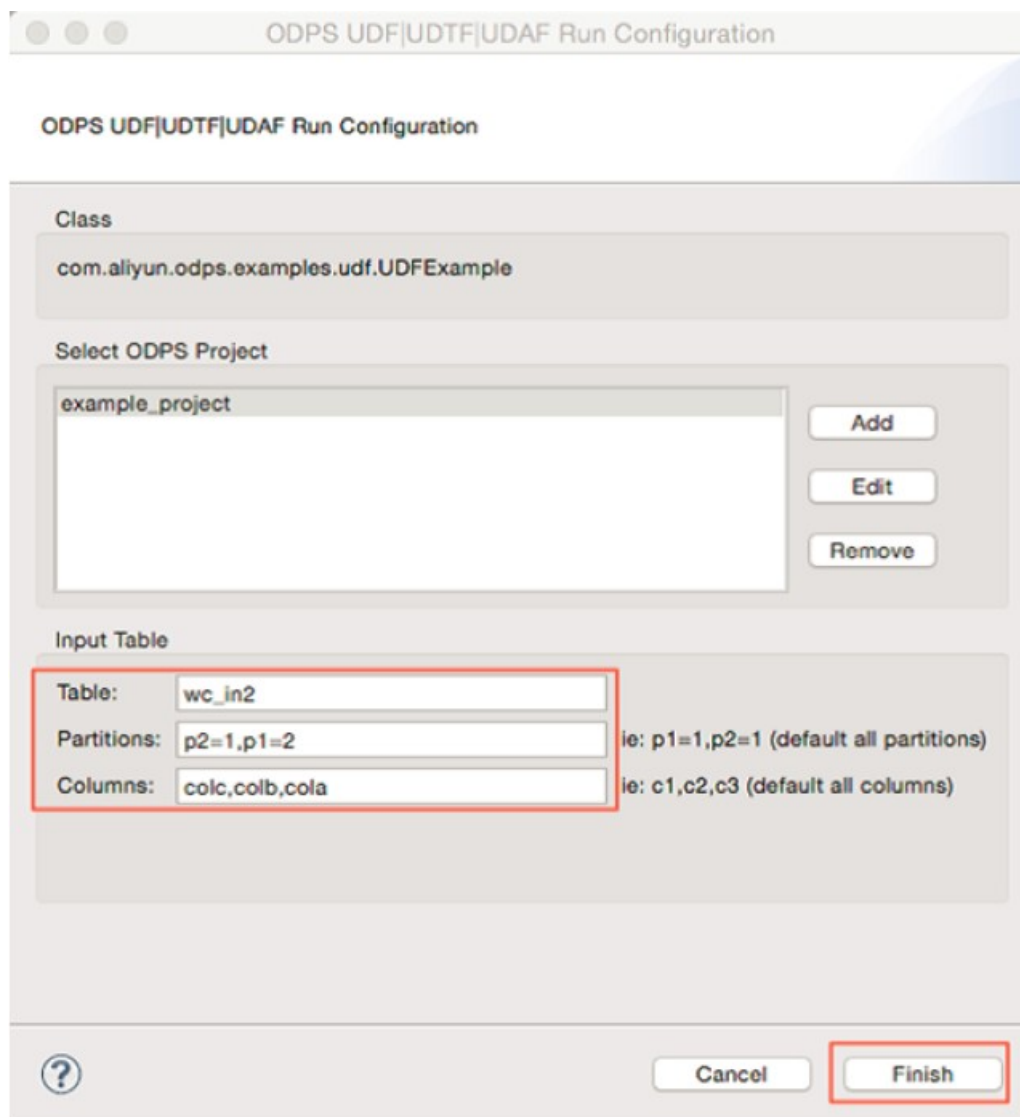
1. Right-click a `udf.java` file (such as `UDFExample.java`) and choose **Run As > Run UDF|UDAF|UDTF** from the shortcut menu, as shown in the following figure.

Step 1



2. In the dialog box that appears, configure the relevant parameters, as shown in the following figure.

Step 2



Note Table indicates the input table of the UDF. Partitions indicate the partitions from which data is read. Multiple partitions are separated by commas. Columns indicate the columns from which data is read. Multiple columns are separated by commas. These parameters are imported to the UDF as parameters.

3. Click **Finish** to run the UDF and obtain the result.

21.2.4.2. Run a UDF program

This topic describes how to run a UDF program in the Eclipse development plugin.

Procedure

1. Right-click a project and choose **New > UDF** (or choose **File > New > UDF** from the menu bar). 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 as follows:

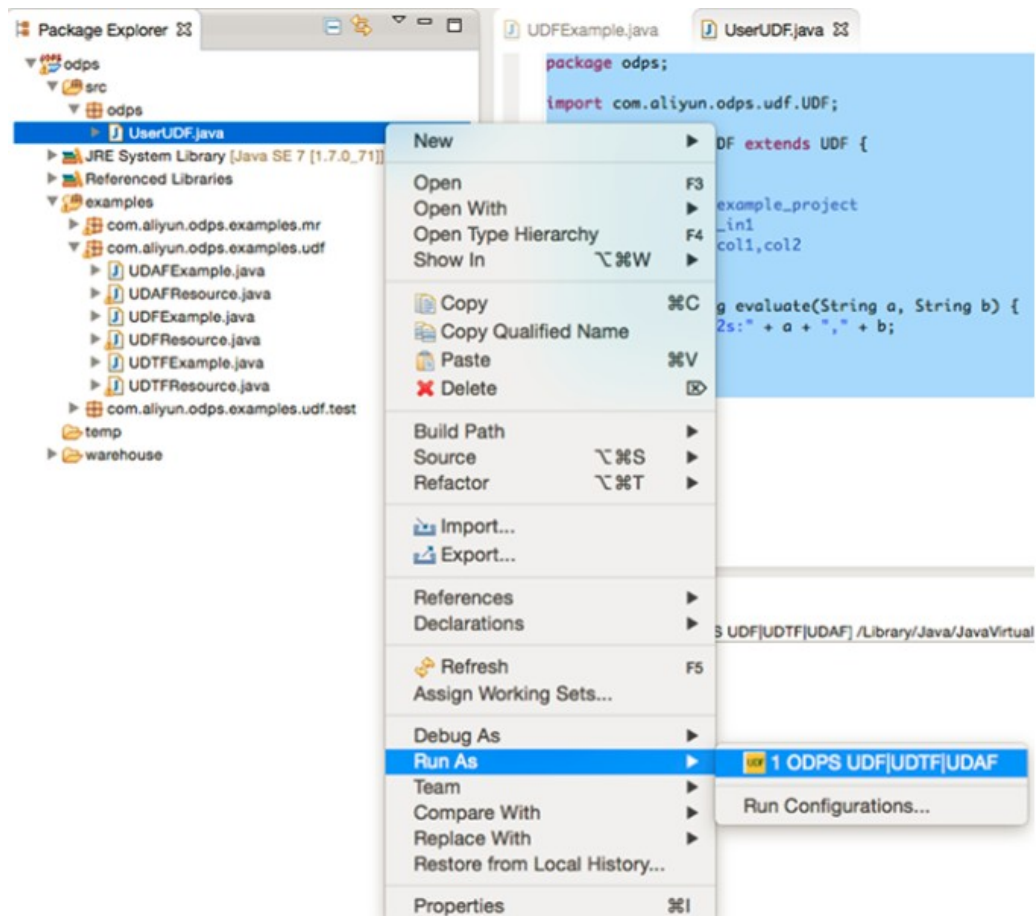

```

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;
}
}

```

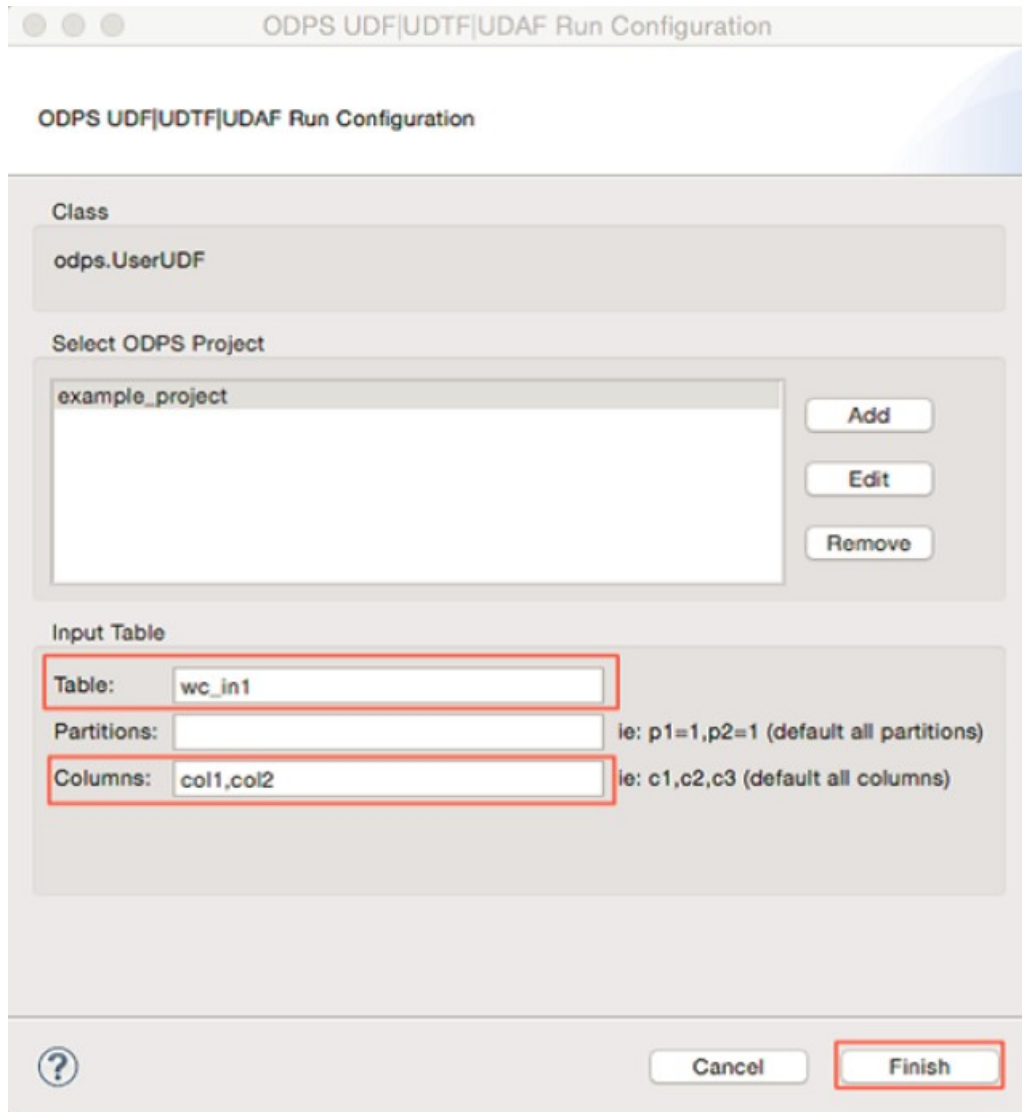
- Right-click the Java file (such as UserUDF.java) and choose **Run As > ODPS UDF|UDTF|UDAF** from the short cut menu, as shown in the following figure.

Step 1



- In the dialog box that appears, configure the relevant parameters, as shown in the following figure.

Step 3-1



The image shows the 'ODPS UDF|UDTF|UDAF Run Configuration' dialog box. It has a title bar with three window control buttons and the text 'ODPS UDF|UDTF|UDAF Run Configuration'. The main area is divided into sections: 'Class' with a text field containing 'odps.UserUDF'; 'Select ODPS Project' with a list box containing 'example_project' and buttons 'Add', 'Edit', and 'Remove'; 'Input Table' with three rows: 'Table: wc_in1', 'Partitions: ' (with a hint 'ie: p1=1,p2=1 (default all partitions)'), and 'Columns: col1,col2' (with a hint 'ie: c1,c2,c3 (default all columns)'). At the bottom, there is a help icon, a 'Cancel' button, and a 'Finish' button. Red boxes highlight the 'Table' and 'Columns' fields, and the 'Finish' button.

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 program. You can use the same method to run a UDTF program.

21.2.5. Graph running example

After creating a MaxCompute project, you can write your own Graph program and debug it locally by performing the following steps.

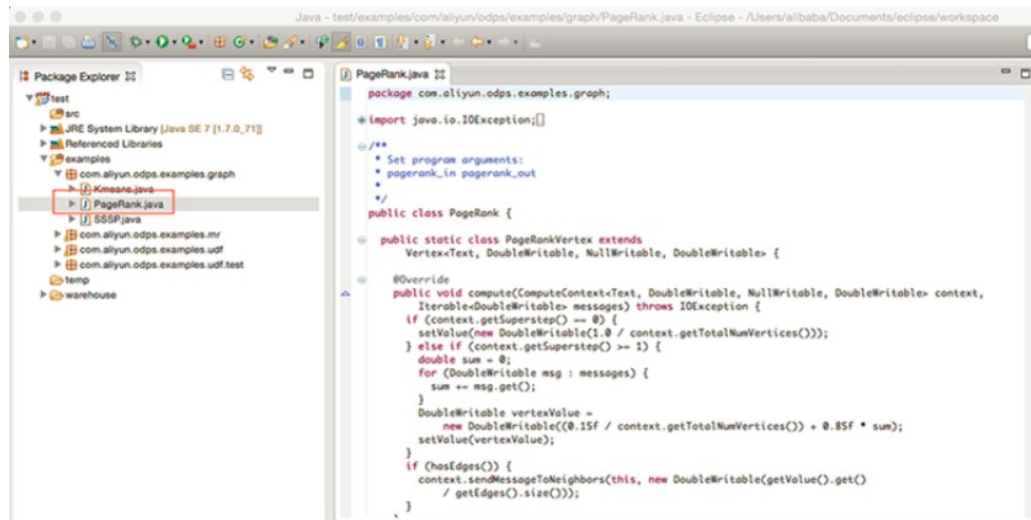
Context

In this example, you can use PageRank.java provided by the plugin to perform local debugging.

Procedure

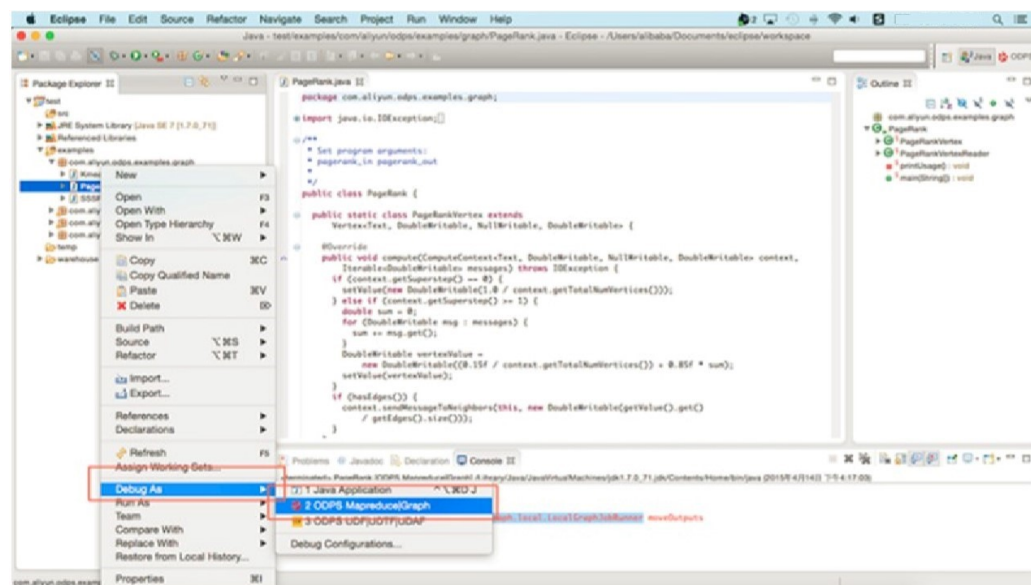
1. Choose **examples > PageRank.java**, as shown in the following figure.

PageRank.java code 1



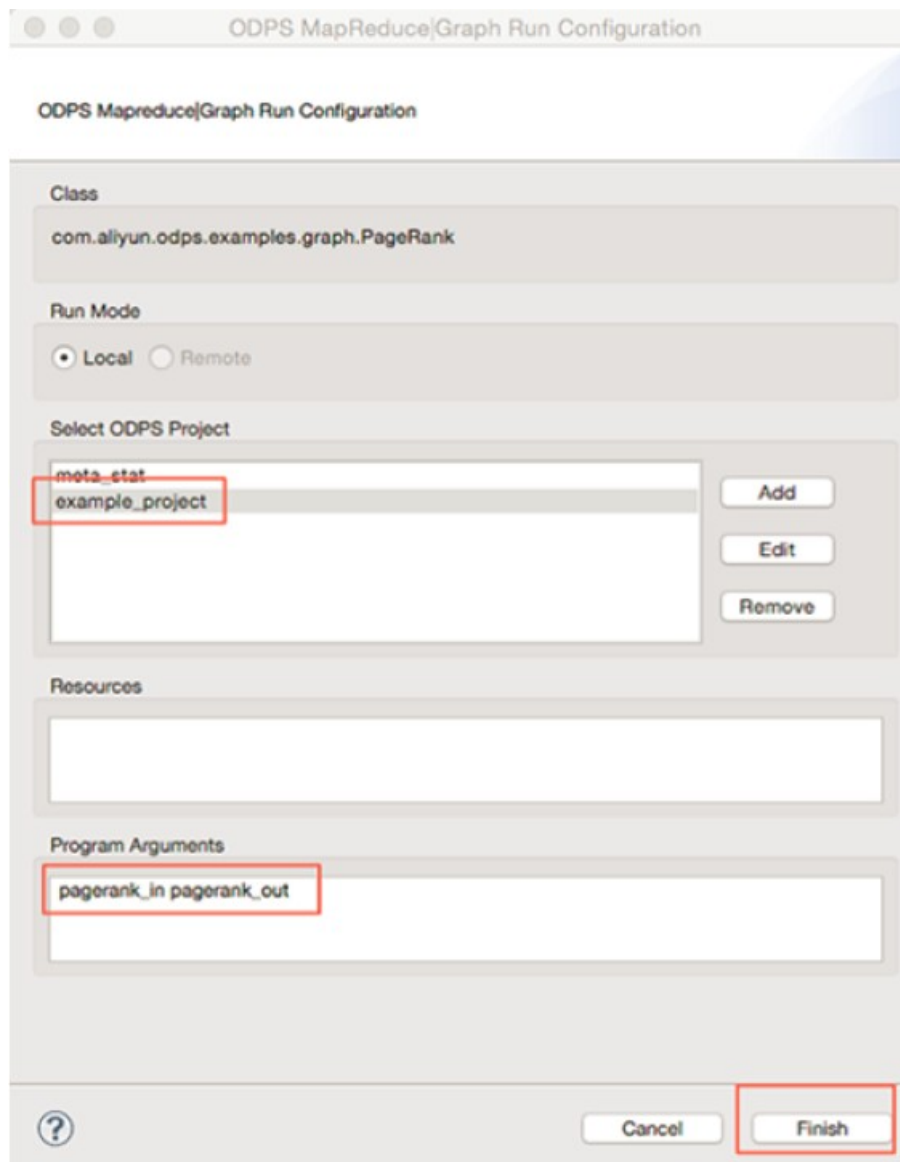
2. Right-click it and choose **Debug As > ODPS MapReduce|Graph**, as shown in the following figure.

PageRank.java code 2



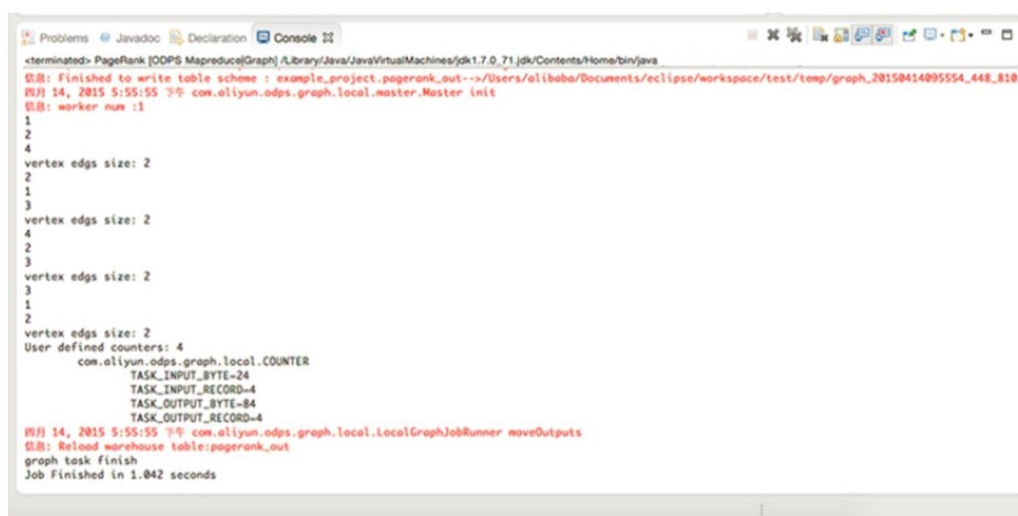
3. In the pop-up dialog box, enter the information as shown below.

Configuration Drawings



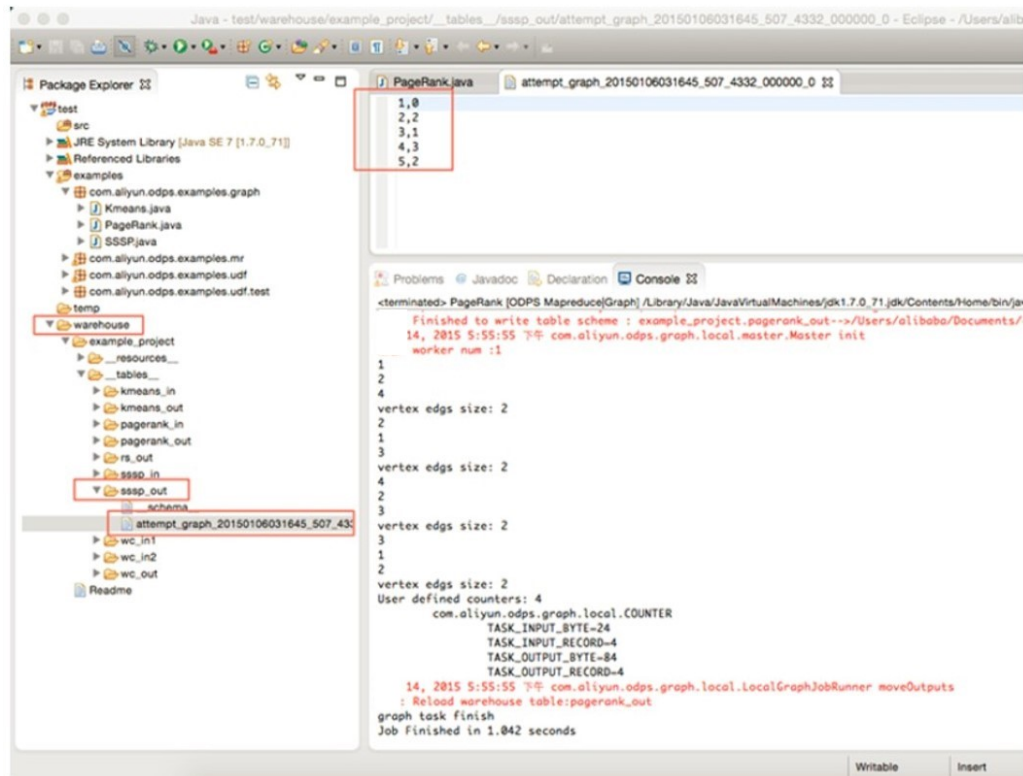
4. Click Finish. Check the running result, as shown in the following figure.

Running Result



Check the local computing result as shown below:

Local computing result



After passing the debugging, you can package the program, upload it to MaxCompute in the form of JAR resource, and submit the 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 uploading JAR resources, see [Compile and run a Graph job](#).

22. MaxCompute FAQ

This topic describes MaxCompute FAQ and solutions.

SQL statement execution is slow. How do I check MaxCompute resource usage?

Log on to the MaxCompute AG as the admin user and perform the following steps:

1. Run the following command to display the remaining resources on the hosts in the MaxCompute cluster in ascending order:

```
r tfrl|sed 's/,//g'|sort -t "|" -k2 -n
```

2. Run the following command to view resource details of the hosts and the total cluster resources in MaxCompute:

```
r ttrl|sed 's/,//g'
```

3. Determine whether the remaining resources in MaxCompute are sufficient based on the ratio of remaining resources to total resources.

How do I handle the slow execution of jobs submitted by a project in a MaxCompute cluster with sufficient resources?

A possible cause is that the resources for the quota group where the project is located 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 MaxCompute AG as the admin user and run the following command to check the resource usage of the quota group:

```
r quota
```

2. If you confirm that the resources for the quota group are exhausted, you can make modifications to the quota list on MaxCompute in Big Data Manager.

How do I modify quota group settings?

1. Run the following command in the MaxCompute AG to create or modify a quota:

```
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 already exists, that quota is modified. Otherwise, a quota with that ID is created.

2. Log on to MaxCompute in Big Data Manager and configure relevant settings.

How do I perform simple operations on the metadata warehouse?

1. Log on to the MaxCompute AG.
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 (package+view)?

You need to install the metadata warehouse enhancement package **package+view** first. Before you install the package, ensure that you are the owner of a project that is granted the package installation permission. After you have installed the package, you can follow the instructions provided in [Package usage](#) to use the smart metadata warehouse.

The following is a brief description on how to use the smart metadata warehouse:

• Installation

package+view is a metadata warehouse enhancement package that depends on the metadata warehouse. You can install it by running the `odpscmd --config=odps_config.ini -f init.sql` command in the *system metadata warehouse* directory.

Note

- If the system asks you to re-install the package, remove the create package comment in the first line of `init.sql` and then run the preceding command again.
- **odps_config.ini**: This configuration file contains the configuration of the account used to access the metadata warehouse, which is also the project owner of the metadata warehouse.

• Authorization

Run the following command to allow a project, such as `p1`, to install the package+view package:

```
odpscmd --config=odps_config.ini -e "allow project p1 to install package systables;"
```

Run the following command to allow all projects to install the package+view package:

```
odpscmd --config=odps_config.ini -e "allow project * to install package systables;"
```

• User operations

You can run the following command to install the package. Ensure 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 see the description of views in the package:

```
desc package meta.systables;
```

 **Note** After you have completed the preceding operations, you can start to use the smart metadata warehouse.

• Views

 **Note** To learn the definition of table schema, refer to the relevant content in the view description, which is available 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	Table, UDF, resource authorization information
package_resources	Object information contained in the package
partitions	Partition information of a partitioned table
policies	User, role, and permission information defined in policies
resources	Resource information
roles	Role information
table_label_grants	Label authorization information of a table
table_labels	Label information of a table
tables	Table and view information
tasks	Job execution records
tunnels	Data upload and download records
udf_resources	Information about resources used in UDFs
udfs	Information about UDFs
user_roles	User and role association information

View name	Content
users	User information

- **Notes**

- By default, the smart metadata warehouse allows you to query data from the past 180 days. If you do not run a job on a specific day, you cannot obtain query results of that day from the smart metadata warehouse.
- We recommend that you specify a query range so that the system does not scan all data from the past 180 days.
- The time the data of the previous day is available depends on the specific output time of the metadata warehouse in the early morning. The data is available immediately after it is generated.
- The metadata warehouse does not provide metadata on the day when the project is created. The purpose is to avoid obtaining data of the project with the same name.

How do I grant Java sandbox permissions?

1. Log on to the AdminConsole and choose **MaxCompute Configuration > Project Management**. Select the project to which you want to grant Java sandbox permissions and double-click it to open the property dialog box.
2. In the **ODPS Sandbox Setting** area, enter the method or class you want to use in Sandbox Java Permissions.

 **Note** Make sure that your input is in the correct format. The following example is for demonstration only. Enter each item in a single line and end it with a semicolon (;).

```
permission java.lang.RuntimePermission "readSystemProperty"; permission java.lang.RuntimePermission "modifyThreadGroup"; permission java.security.AllPermission;
```

3. Click **Finish Modification**.

How do I handle exhausted disk capacity?

In most cases, you can clear scripts to free disk capacity. The most possible cause is that the root directory of MaxCompute AG or the /apsara directory occupies too much disk space. Therefore, you need to clear scripts in these two directories.

How do I find an AccessKey pair and configure correct AccessKey information?

1. Access the framework cluster management page of the Apsara Stack data center, and choose **Operations > Cluster Operations**. Select the cluster for which you want to find and configure the AccessKey pair, and access the cluster configuration page.
2. Double-click the kv.conf file in the file list to find the AccessKey information. You can also modify the information in this file and then save your modifications.

How do I add a MaxCompute host to a blacklist?

1. Log on to the Apsara AG as the admin user. Run the following command to enable the Fuxi blacklist 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 a MaxCompute host to the Fuxi blacklist:

```
/apsara/deploy/rpc_wrapper/rpc.shblacklist cluster add $hostname
```

4. Run the following command to view the Fuxi blacklist again and confirm that the host is added:

```
/apsara/deploy/rpc_wrapper/rpc.shblacklist cluster get
```

How do I export data from MaxCompute?

There are two methods to export data from MaxCompute: The first is to use the Tunnel command. The second is to configure synchronization tasks in DataWorks to export data from MaxCompute to other destinations.

How do I view the current MaxCompute version?

Run the following commands to view the MaxCompute version:

```
cat /apsara/odps_info/version|grep odps
```

```
cat /apsara/version
```

How do I restart MaxCompute services?

1. Run the following commands to save the configurations of resident MaxCompute services 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 corresponding command.
You can use 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 MaxCompute services:

```

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 commands to start MaxCompute services:

```

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 power MaxCompute on and off?

1. Run the following commands to save the configurations of resident MaxCompute services 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 corresponding command.
You can use the r al command to view resident services.
r plan Odps/CGServiceControllerx > CGServiceControllerx
r plan sys/sqlonline-OTS > sqlonline-OTS
r plan Odps/MessengerServicex > MessengerServicex
r plan Odps/OdpsServicex > OdpsServicex
r plan Odps/HiveServerx > HiveServerx
r plan Odps/XStreamServicex > XStreamServicex
r plan Odps/QuotaServicex > QuotaServicex
r plan Odps/ReplicationServicex > ReplicationServicex

```

2. Run the following commands to stop MaxCompute services:

```

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 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 system:

```
/home/admin/dayu/bin/allapsara start
```

7. Run the following commands to start MaxCompute services:

```
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 too many resources.
2. Generally, resources are occupied by user tasks. If task processes occupy too many resources, wait until these tasks are completed or ask users to stop these tasks.

23. Open source features of MaxCompute

This topic describes open source features related to MaxCompute.

SDKs

MaxCompute provides Java SDK and Python SDK interfaces to create, view, and delete MaxCompute tables. You can use the SDKs to manage MaxCompute by editing code.

How to obtain service support: Visit the official documentation or submit a ticket online.

MaxCompute RODPS

MaxCompute RODPS is an R plug-in for MaxCompute. For more information about the plug-in, see [ODPS Plugin for R](#) on GitHub.

How to obtain service support: Leave a message or create an issue in [ODPS Plugin for R](#) on GitHub.

MaxCompute JDBC

MaxCompute JDBC is an official JDBC driver provided by MaxCompute. It provides a set of interfaces to execute SQL tasks for Java programs. The project is hosted in [ODPS JDBC](#) on GitHub.

How to obtain service support: Leave a message or create an issue in [ODPS JDBC](#) on GitHub.

Mars

Mars is a tensor-based unified distributed computing framework. Mars makes it possible to execute large-scale scientific computing tasks by using only several lines of code, whereas MapReduce requires hundreds of lines of code. In addition, Mars improves computing performance.

The source code of Mars is now available on GitHub. You are welcome to contribute to Mars. You can visit [Mars](#) on GitHub to obtain its open source code.

For more information about Mars, see [Mars Development Guide](#).

How to obtain service support: Leave a message or create an issue in [Mars](#) 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.

The Flume and OGG plug-ins are implemented based on the DataHub SDK, whereas Sqoop, the 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, the Kettle plug-in, and Hive Data Transfer UDTF are used to transfer data in batches 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.

How to obtain service support: Leave a message or create an issue in [Aliyun MaxCompute Data Collectors](#) on GitHub.