ALIBABA CLOUD

Alibaba Cloud

Apsara Stack Enterprise

Developer Guide - Analytics and Artificial Intelligence

Product Version: 2006, Internal: V3.12.0

Document Version: 20201112

Alibaba Cloud

Legal disclaimer

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

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

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

Table of Contents

1.MaxCompute 27
1.1. Java SDK 27
1.1.1. SDK for Java 27
1.1.2. Prerequisites 28
1.1.2.1. Overview 28
1.1.2.2. Obtain an AccessKey pair 28
1.1.2.2.1. Log on to the ASCM console 28
1.1.2.2.2. Obtain the AccessKey pair of an organization 29
1.1.2.2.3. View the AccessKey pair of your Apsara Stack te... 29
1.1.2.3. Obtain the endpoint 29
1.1.3. Major APIs 31
1.1.3.1. AliyunAccount 31
1.1.3.2. Odps 32
1.1.3.3. Projects 32
1.1.3.4. Project 32
1.1.3.5. SQLTask 33
1.1.3.6. Instances 33
1.1.3.7. Instance 34
1.1.3.8. Tables 35
1.1.3.9. Table 36
1.1.3.10. Resources 36
1.1.3.11. Resource 37
1.1.3.12. Functions 38
1.1.3.13. Function 38
1.1.3.14. Spark Shell 39
1.1.3.15. Spark R 39

1.1.3.16. Spark SQL 40
1.1.3.17. Spark JDBC 40
1.1.3.18. Document APIs 41
1.1.3.19. Search APIs 41
1.2. SDK for Python 41
2.DataWorks 57
2.1. Overview 57
2.2. Terms 57
2.3. List of operations by function 57
2.4. Make API requests 58
2.4.1. Java SDK 58
2.4.2. HTTP API 62
2.4.2.1. Endpoints 62
2.4.2.2. Protocol 62
2.4.2.3. Request method 62
2.4.2.4. Encoding 63
2.4.2.5. Signature method 63
2.4.2.6. Response parameters 63
2.4.2.7. API example 64
2.5. APIs for HTTP requests 66
2.5.1. Create a node 66
2.5.2. Update a node 68
2.5.3. Update multiple nodes at a time 69
2.5.4. Delete a node 72
2.5.5. Query parent nodes 72
2.5.6. Query child nodes 73
2.5.7. Query the node code 74
2.5.8. Query nodes 75

2.5.9. Query a node by ID 78
2.5.10. Obtain the total number of nodes 79
2.5.11. Query node instances 80
2.5.12. Query the operational logs of a node instance 83
2.5.13. Query a node instance by ID 84
2.5.14. Query the parent node instance by level 85
2.5.15. Query the child node instances at a specified level 86
2.5.16. Query the number of node instances in each status o... 87
2.5.17. Query the number of task instances in each status on... 88
2.5.18. Query the number of node instances in each status f... 89
2.5.19. Rerun an instance 92
2.5.20. Restore an instance 93
2.5.21. Set the instance status to Success and trigger descen... 93
2.5.22. Terminate a task 94
2.5.23. Rerun a descendant instance 95
2.5.24. Update the scheduling configurations of an instance 96
2.5.25. Pause an instance 96
2.5.26. Terminate multiple instances 97
2.5.27. Rerun multiple instances 98
2.5.28. Resume multiple tasks 98
2.5.29. Pause multiple instances 99
2.5.30. Set the status of multiple instances to Success 100
2.5.31. Retrieve the total number of instances on the day 101
2.5.32. Query business flow instances 101
2.5.33. Retroactive execution 102
2.5.34. Run a smoke test 103
2.5.35. Execute an ad-hoc business flow 104
2.5.36. Terminate a business flow 105

2.5.37. Submit a resource package 106
2.5.38. Query the submission status of a resource package 107
2.5.39. Data structure 107
2.5.39.1. NodeModifyDto 107
2.5.39.2. TaskEntity 109
2.5.39.3. DagEntity 112
2.5.39.4. TaskStatus 113
2.5.39.5. Dagstatus 113
2.5.39.6. ProgramType 113
2.5.39.7. CycleType 114
2.5.39.8. DependencyType 114
2.6. Java SDK 114
2.6.1. GetInstanceSummary 114
2.6.2. AddResGroupGateWay 116
2.6.3. CreateConnection 117
2.6.4. CreateDag 119
2.6.5. CreateDQCEntity 121
2.6.6. CreateDQCFollower 123
2.6.7. CreateDQCRule 125
2.6.8. CreateManualDag 127
2.6.9. CreateMetaSpiderJob 128
2.6.10. CreateResGroup 129
2.6.11. DeleteConnections 131
2.6.12. DeleteDQCEntity 132
2.6.13. DeleteDQCFollower 133
2.6.14. DeleteDQCRule 134
2.6.15. DeleteProjectResGroup 135
2.6.16. DeleteResGroupGateWay 136

2.6.17. GetDagDetail 137
2.6.18. GetDataServiceApiDetail 139
2.6.19. GetDataServiceAppDetail 140
2.6.20. GetDefaultTenant 140
2.6.21. GetDQCEntity 141
2.6.22. GetDQCFollower 144
2.6.23. GetDQCRule 145
2.6.24. GetNodeDetail 150
2.6.25. GetNodeUpdateStatus 156
2.6.26. GetResGroupAk 158
2.6.27. GetResGroupGatewayList 160
2.6.28. GetResGroupList 161
2.6.29. GetTableColumn 166
2.6.30. GetTableList 167
2.6.31. GetTaskLog 169
2.6.32. GetUserInfo 171
2.6.33. ListConnection 172
2.6.34. ListDataServiceApps 178
2.6.35. ListDataServiceAuthedApi 178
2.6.36. ListPermission 179
2.6.37. ListProjectModule 180
2.6.38. ListProjectModules 181
2.6.39. ListProject 182
2.6.40. ListTenantModule 182
2.6.41. ListUserPermission 183
2.6.42. ModifyBusiness 183
2.6.43. ModifyNode 185
2.6.44. ModifySolution 188

2.6.45. RerunTask 189
2.6.46. ResumeTask 191
2.6.47. SearchBusiness 192
2.6.48. SearchManualDagNodeInstance 194
2.6.49. SearchSolution 195
2.6.50. SearchTasks 196
2.6.51. TestConnectivity 201
2.6.52. UpdateDQCFollower 202
2.6.53. UpdateDQCRule 204
2.6.54. UpdateResGroupGateWay 206
2.6.55. CheckMetaTable 208
2.6.56. GetMetaDB 208
2.6.57. GetMetaTable 210
2.6.58. GetMetaTableIntroWiki 212
2.6.59. ListMetaColumnLineage 213
2.6.60. ListMetaTableChangeLog 214
2.6.61. ListMetaTableColumn 216
2.6.62. ListMetaTableImpact 217
2.6.63. ListMetaTableIntroWiki 218
2.6.64. ListMetaTableLineage 219
2.6.65. ListMetaTableOutput 220
2.6.66. ListMetaTablePartition 222
2.6.67. SearchMetaTables 223
2.6.68. GetStorageAndCalcMetrics 225
2.6.69. GetInstanceStatistic 226
2.6.70. GetInstanceLog 227
2.6.71. GetNodeCode 228
2.6.72. QueryInstances 229

2.7. Obtain an AccessKey pair 232
3.Realtime Compute 234
3.1. Flink SQL 234
3.1.1. Overview 234
3.1.2. Keywords 234
3.1.3. Terms 236
3.1.3.1. Watermark 236
3.1.3.2. Computed column 238
3.1.3.3. Time attributes 239
3.1.4. Data types 242
3.1.4.1. Data types 242
3.1.4.2. Data types and operators 243
3.1.5. DDL statements 244
3.1.5.1. Overview 244
3.1.5.2. Create a data source table 248
3.1.5.2.1. Overview 248
3.1.5.2.2. Create a DataHub source table 250
3.1.5.2.3. Create a Log Service source table 254
3.1.5.3. Create a data result table 259
3.1.5.3.1. Overview 259
3.1.5.3.2. Create a DataHub result table 260
3.1.5.3.3. Create a Log Service result table 262
3.1.5.3.4. Create a Table Store result table 263
3.1.5.3.5. Create an RDS result table 265
3.1.5.4. Create a data dimension table 270
3.1.5.4.1. Overview 270
3.1.5.4.2. Create a Table Store dimension table 271
3.1.5.4.3. Create an RDS dimension table 273

3.1.6. DML statements 277
3.1.6.1. INSERT INTO statement 277
3.1.7. QUERY statements 278
3.1.7.1. SELECT statement 278
3.1.7.2. WHERE clause 280
3.1.7.3. HAVING clause 281
3.1.7.4. GROUP BY clause 282
3.1.7.5. JOIN statement 283
3.1.7.6. JOIN statement for dimension tables 286
3.1.7.7. UNION ALL clause 289
3.1.7.8. Top_N statement 292
3.1.7.9. Complex event processing (CEP) 298
3.1.8. Create a view 305
3.1.9. Window functions 308
3.1.9.1. Overview 308
3.1.9.2. Session window 313
3.1.9.3. Sliding window 317
3.1.9.4. Tumbling window 320
3.1.9.5. Over window 323
3.1.10. Logical operations 331
3.1.10.1. = 331
3.1.10.2. > 332
3.1.10.3. >= 333
3.1.10.4. < 333
3.1.10.5. <= 334
3.1.10.6. <> 335
3.1.10.7. AND 336
3.1.10.8. BETWEEN AND 337

3.1.10.9. IS NOT FALSE 338
3.1.10.10. IS NOT NULL 339
3.1.10.11. IS NOT TRUE 340
3.1.10.12. IS NOT UNKNOWN 340
3.1.10.13. IS NULL 341
3.1.10.14. IS TRUE 342
3.1.10.15. IS UNKNOWN 343
3.1.10.16. LIKE 344
3.1.10.17. NOT 345
3.1.10.18. NOT BETWEEN AND 346
3.1.10.19. OR 347
3.1.10.20. IN 348
3.1.10.21. NOT IN 349
3.1.10.22. IS DISTINCT FROM 350
3.1.10.23. IS NOT DISTINCT FROM 350
3.1.11. Built-in functions 351
3.1.11.1. String functions 351
3.1.11.1.1. Overview 351
3.1.11.1.2. CHAR_LENGTH 357
3.1.11.1.3. CHR 358
3.1.11.1.4. CONCAT 359
3.1.11.1.5. CONCAT_WS 360
3.1.11.1.6. FROM_BASE64 361
3.1.11.1.7. HASH_CODE 361
3.1.11.1.8. INITCAP 362
3.1.11.1.9. JSON_VALUE 363
3.1.11.1.10. KEYVALUE 364
3.1.11.1.11. LOWER 366

3.1.11.1.12. LPAD 366
3.1.11.1.13. MD5 368
3.1.11.1.14. OVERLAY 369
3.1.11.1.15. PARSE_URL 369
3.1.11.1.16. POSITION 370
3.1.11.1.17. REGEXP 371
3.1.11.1.18. REGEXP_EXTRACT 372
3.1.11.1.19. REGEXP_REPLACE 373
3.1.11.1.20. REPEAT 375
3.1.11.1.21. REVERSE 376
3.1.11.1.22. RPAD 376
3.1.11.1.23. SPLIT_INDEX 378
3.1.11.1.24. STR_TO_MAP 379
3.1.11.1.25. SUBSTRING 380
3.1.11.1.26. TO_BASE64 381
3.1.11.1.27. TRIM 382
3.1.11.1.28. UPPER 382
3.1.11.2. Mathematical unctions 383
3.1.11.2.1. Addition 383
3.1.11.2.2. Subtraction 384
3.1.11.2.3. Multiplication 385
3.1.11.2.4. Division 385
3.1.11.2.5. ABS 386
3.1.11.2.6. ACOS 387
3.1.11.2.7. BIN 387
3.1.11.2.8. ASIN 388
3.1.11.2.9. ATAN 389
3.1.11.2.10. BITAND 390

3.1.11.2.11. BITNOT 391
3.1.11.2.12. BITOR 391
3.1.11.2.13. BITXOR 392
3.1.11.2.14. CARDINALITY 393
3.1.11.2.15. COS 393
3.1.11.2.16. COT 394
3.1.11.2.17. E 395
3.1.11.2.18. EXP 396
3.1.11.2.19. FLOOR 396
3.1.11.2.20. LN 397
3.1.11.2.21. LOG 398
3.1.11.2.22. LOG10 399
3.1.11.2.23. LOG2 400
3.1.11.2.24. PI 400
3.1.11.2.25. POWER 401
3.1.11.2.26. RAND 402
3.1.11.2.27. SIN 402
3.1.11.2.28. SQRT 403
3.1.11.2.29. TAN 404
3.1.11.2.30. CEIL 404
3.1.11.2.31. CHARACTER_LENGTH 405
3.1.11.2.32. DEGREES 406
3.1.11.2.33. MOD 406
3.1.11.2.34. ROUND 407
3.1.11.3. Date functions 408
3.1.11.3.1. CURRENT_TIMESTAMP 408
3.1.11.3.2. DATEDIFF 409
3.1.11.3.3. DATE_ADD 410

3.1.11.3.4. DATE_FORMAT 411
3.1.11.3.5. DATE_SUB 412
3.1.11.3.6. DAYOFMONTH 412
3.1.11.3.7. FROM_UNIXTIME 413
3.1.11.3.8. HOUR 414
3.1.11.3.9. LOCALTIME 415
3.1.11.3.10. MINUTE 416
3.1.11.3.11. MONTH 416
3.1.11.3.12. NOW 417
3.1.11.3.13. SECOND 418
3.1.11.3.14. TIMESTAMPADD 419
3.1.11.3.15. TO_DATE 420
3.1.11.3.16. TO_TIMESTAMP 421
3.1.11.3.17. UNIX_TIMESTAMP 422
3.1.11.3.18. WEEK 423
3.1.11.3.19. YEAR 424
3.1.11.4. Conditional functions 424
3.1.11.4.1. CASE WHEN 424
3.1.11.4.2. COALESCE 426
3.1.11.4.3. IF 427
3.1.11.4.4. IS_ALPHA 428
3.1.11.4.5. IS_DECIMAL 429
3.1.11.4.6. IS_DIGIT 430
3.1.11.4.7. NULLIF 430
3.1.11.5. Table-valued functions 431
3.1.11.5.1. GENERATE_SERIES 431
3.1.11.5.2. JSON_TUPLE 432
3.1.11.5.3. STRING_SPLIT 433

3.1.11.6. Type conversion functions 434
3.1.11.6.1. CAST 434
3.1.11.7. Aggregate functions 435
3.1.11.7.1. AVG 435
3.1.11.7.2. CONCAT_AGG 436
3.1.11.7.3. COUNT 437
3.1.11.7.4. FIRST_VALUE 437
3.1.11.7.5. LAST_VALUE 439
3.1.11.7.6. MAX 440
3.1.11.7.7. MIN 441
3.1.11.7.8. STDDEV_POP 442
3.1.11.7.9. SUM 443
3.1.11.7.10. VAR_POP 444
3.1.11.8. Other functions 445
3.1.11.8.1. UUID 445
3.1.12. UDXs 445
3.1.12.1. Overview 446
3.1.12.2. User-defined scalar function (UDSF) 448
3.1.12.3. User-defined aggregation function (UDAF) 450
3.1.12.4. User-defined table function (UDTF) 454
3.1.12.5. Create a user-defined function (UDF) by using Inte... 460
4.E-MapReduce (EMR) 469
4.1. Prerequisites 469
4.1.1. Development prerequisites 469
4.1.2. Configure the OSS URI to use EMR 469
4.1.3. Use sample projects 470
4.1.3.1. Sample project 470
4.1.3.2. Dependencies 471

4.1.3.3. Preparations 471
4.1.3.4. Concepts 471
4.1.3.5. Cluster-based running 471
4.1.3.6. Local running 474
4.1.3.6.1. IntelliJ IDEA 474
4.1.3.6.2. Scala IDE for Eclipse 476
4.2. Hadoop development 479
4.2.1. Parameter description 479
4.2.2. Create and run MapReduce jobs 481
4.2.2.1. Process OSS data in MapReduce 481
4.2.2.2. Word count 481
4.2.2.3. Use Maven to manage MapReduce jobs 484
4.2.3. Create and run a Hive job 490
4.2.3.1. Use OSS in Hive 490
4.2.3.2. Use Tez as the computing engine 491
4.2.4. Create and run a Pig job 493
4.2.5. Create a Hadoop streaming job 496
4.2.6. Process Table Store data in Hive 497
4.2.6.1. Connect Table Store to Hive 497
4.2.6.2. Data type conversion 498
4.2.6.3. Appendix 499
4.2.7. Process Table Store data in MapReduce 499
4.2.7.1. Connect Table Store to MapReduce 499
4.2.7.2. Appendix 502
4.3. Create an HBase cluster and use the HBase storage service 503
4.3.1. Create an HBase cluster and use the HBase storage ser... 503
4.3.2. HBase configuration 503
4.3.3. Access HBase 504

4.3.4. Example 505
4.3.5. HBase backup 506
4.4. Data transmission application - Sqoop 507
4.4.1. Install Sqoop 507
4.4.2. Data transmission 508
4.4.2.1. Import data from MySQL to HDFS 508
4.4.2.2. Import data from HDFS to MySQL 509
4.4.2.3. Import data from MySQL to Hive 509
4.4.2.4. Import data from Hive to MySQL 510
4.4.2.5. Import data from MySQL to OSS 510
4.4.2.6. Import data from OSS to MySQL 511
4.4.2.7. Use SQL to import data 511
5.DataHub 513
5.1. SDK overview 513
5.2. Prerequisites 513
5.2.1. Overview 513
5.2.2. Obtain an AccessKey pair 513
5.2.2.1. Log on to the ASCM console 513
5.2.2.2. Obtain the AccessKey pair of an organization 514
5.2.2.3. View the AccessKey pair of your Apsara Stack tena... 514
5.2.3. Obtain the endpoint 515
5.3. DataHub SDK for Java 517
5.3.1. Overview 517
5.3.2. Preparations 517
5.3.3. Ingest tuple data 518
5.3.3.1. Create a tuple topic 518
5.3.3.2. Obtain the list of shards 518
5.3.3.3. Write data into DataHub 518

5.3.3.4. Consume data 519
5.3.4. Ingest blob data 520
5.3.4.1. Create a blob topic 520
5.3.4.2. Obtain the list of shards 521
5.3.4.3. Write data into DataHub 521
5.3.4.4. Consume data 521
5.3.5. Data ingestion methods 522
5.3.5.1. Ingest data based on shard IDs 522
5.3.5.2. Ingest data based on hash values 523
5.3.5.3. Ingest data based on partition keys 524
5.3.6. Create a DataConnector 525
5.3.6.1. Create a DataConnector for MaxCompute 525
5.3.6.2. Create a DataConnector for AnalyticDB or ApsaraDB.. 526
5.4. DataHub SDK for Python 528
5.4.1. Install DataHub SDK for Python 528
5.4.2. Preparations 528
5.4.3. Create topics 528
5.4.4. Write records into DataHub and subscribe applications... 530
5.5. Scalability 533
5.5.1. Scenarios 533
5.5.2. Obtain shard details 533
5.5.3. Split a shard 534
5.5.4. Merge shards 534
5.6. Insert a column to a topic schema 535
5.6.1. Insert a column 535
6.Graph Analytics 537
6.1. List of operations by function 537
6.2. Business restrictions 539

6.3. Make API requests 540
6.3.1. Request structures 540
6.3.2. Common parameters 540
6.3.3. Responses 541
6.3.4. API restrictions 543
6.3.5. Authentication 543
6.4. Search 544
6.4.1. QueryLinks 544
6.4.2. QueryNodes 547
6.5. Relationship network 549
6.5.1. GetFastGraph 549
6.5.2. GetGraph 554
6.5.3. GetGroupGraph 561
6.5.4. CommonLinkGraph 568
6.5.5. PathAnalysisGraph 575
6.5.6. KinshipGraph 582
6.6. Data persistence 586
6.6.1. PersistNode 586
6.6.2. PersistLink 589
6.7. OLEP escape 593
6.7.1. QueryOlpMetaConfig 593
6.7.2. UpdateOlpMetaConfig 596
6.8. ShareAnalysis 598
6.9. Labels 599
6.9.1. AddLabels 599
6.9.2. ObtainObjectLabels 602
6.9.3. ObtainLabelHistory 603
6.9.4. LikeLabels 605

6.9.5. CancelLikes 607
6.9.6. ModifyLabels 608
6.9.7. DeleteLabels 611
6.10. Pattern recognition 612
6.10.1. ObtainPatternsandDirectories 612
6.10.2. CreateTasks 615
6.10.3. QueryTaskStatus 615
6.10.4. ViewTaskResults 616
6.11. Terms 617
7.Elasticsearch (on ECS) 619
7.1. List of operations by function 619
7.2. RESTful API 621
7.2.1. Single-document operations 621
7.2.1.1. Index operation 621
7.2.1.1.1. Automatic index creation 622
7.2.1.1.2. Versioning 623
7.2.1.1.3. Operation type 624
7.2.1.1.4. Automatic ID generation 625
7.2.1.1.5. Routing 625
7.2.1.1.6. Distribution 626
7.2.1.1.7. Wait for active shards 626
7.2.1.1.8. Refresh 627
7.2.1.1.9. Noop updates 627
7.2.1.1.10. Timeout 627
7.2.1.2. Get operation 628
7.2.1.2.1. Real-time Get operation 629
7.2.1.2.2. Source filtering 629
7.2.1.2.3. Stored fields 629

7.2.1.2.4. Obtain _source 632
7.2.1.2.5. Routing 632
7.2.1.2.6. Preference 633
7.2.1.2.7. Refresh 633
7.2.1.2.8. Distribution 633
7.2.1.2.9. Versioning support 633
7.2.1.3. Delete operation 633
7.2.1.3.1. Versioning 634
7.2.1.3.2. Routing 634
7.2.1.3.3. Automatic index creation 634
7.2.1.3.4. Distribution 635
7.2.1.3.5. Wait for active shards 635
7.2.1.3.6. Refresh 635
7.2.1.3.7. Timeout 635
7.2.1.4. Update operation 635
7.2.1.4.1. Updates by using scripts 636
7.2.1.4.2. Update a part of a document 637
7.2.1.4.3. Detect noop updates 638
7.2.1.4.4. Upsert 639
7.2.1.4.5. Parameters 640
7.2.2. Multi-document operations 641
7.2.2.1. Multi Get operation 641
7.2.2.1.1. Source filtering 642
7.2.2.1.2. Fields 643
7.2.2.1.3. Routing 644
7.2.2.1.4. Security 645
7.2.2.2. Batch operation 645
7.2.2.3. Delete by query 650

7.2.2.3.1. URL parameters 653
7.2.2.3.2. Response body 654
7.2.2.3.3. Cancel tasks 655
7.2.2.3.4. Task operation 655
7.2.2.3.5. Rethrottling 657
7.2.2.3.6. Slicing 657
7.2.2.4. Update by query 661
7.2.2.4.1. URL parameters 664
7.2.2.4.2. Response body 665
7.2.2.4.3. Task operation 666
7.2.2.4.4. Cancel tasks 668
7.2.2.4.5. Rethrottling 668
7.2.2.4.6. Slicing 668
7.2.2.4.7. Add new properties 671
7.2.2.5. Reindex 673
7.2.2.5.1. Reindex from a remote cluster 682
7.2.2.5.2. URL parameters 684
7.2.2.5.3. Response body 685
7.2.2.5.4. Task operation 687
7.2.2.5.5. Cancel tasks 689
7.2.2.5.6. Rethrottling 689
7.2.2.5.7. Reindex to change a field name 689
7.2.2.5.8. Slicing 691
7.2.2.5.9. Reindex daily indexes 693
7.2.2.5.10. Extract a random subset of an index 694
8.Elasticsearch (on k8s) 696
8.1. List of operations by function 696
8.2. RESTful API 698

8.2.1. Single-document operations 698
8.2.1.1. Index 698
8.2.1.1.1. Automatic index creation 699
8.2.1.1.2. Versioning 700
8.2.1.1.3. Operation type 701
8.2.1.1.4. Automatic ID generation 702
8.2.1.1.5. Routing 702
8.2.1.1.6. Distribution 703
8.2.1.1.7. Wait for active shards 703
8.2.1.1.8. Refresh 704
8.2.1.1.9. Noop updates 704
8.2.1.1.10. Timeout 704
8.2.1.2. Get 705
8.2.1.2.1. Real-time get operation 706
8.2.1.2.2. Source filtering 706
8.2.1.2.3. Stored fields 706
8.2.1.2.4. Obtain only the _source field 709
8.2.1.2.5. Routing 709
8.2.1.2.6. Preference 710
8.2.1.2.7. Refresh 710
8.2.1.2.8. Distribution 710
8.2.1.2.9. Versioning support 710
8.2.1.3. Delete 710
8.2.1.3.1. Versioning 711
8.2.1.3.2. Routing 711
8.2.1.3.3. Automatic index creation 711
8.2.1.3.4. Distribution 712
8.2.1.3.5. Wait for active shards 712

8.2.1.3.6. Refresh 712
8.2.1.3.7. Timeout 712
8.2.1.4. Update operation 712
8.2.1.4.1. Perform a scripted update 713
8.2.1.4.2. Perform a partial update 714
8.2.1.4.3. Detect noop updates 715
8.2.1.4.4. Upsert 716
8.2.1.4.5. Parameters 717
8.2.2. Multi-document operations 718
8.2.2.1. Multi get 718
8.2.2.1.1. Source filtering 719
8.2.2.1.2. Fields 720
8.2.2.1.3. Routing 721
8.2.2.1.4. Security 722
8.2.2.2. Bulk 722
8.2.2.3. Delete by query 727
8.2.2.3.1. URL parameters 730
8.2.2.3.2. Response body 731
8.2.2.3.3. Cancel a _delete_by_query task 732
8.2.2.3.4. Obtain the status of running _delete_by_query ... 732
8.2.2.3.5. Rethrottling 734
8.2.2.3.6. Slicing 734
8.2.2.4. Update By Query operation 738
8.2.2.4.1. URL parameters 741
8.2.2.4.2. Response body 742
8.2.2.4.3. Obtain the status of running update-by-query r... 743
8.2.2.4.4. Cancel tasks 745
8.2.2.4.5. Rethrottling 745

8.2.2.4.6. Slicing 745
8.2.2.4.7. Obtain a new property 748
8.2.2.5. Reindex 750
8.2.2.5.1. Reindex from a remote cluster 759
8.2.2.5.2. URL parameters 761
8.2.2.5.3. Response body 762
8.2.2.5.4. Obtain the status of all running reindex reques... 764
8.2.2.5.5. Cancel a reindexing task 766
8.2.2.5.6. Rethrottling 766
8.2.2.5.7. Perform reindexing to change the name of a fie... 766
8.2.2.5.8. Slicing 768
8.2.2.5.9. Reindex daily indexes 771
8.2.2.5.10. Extract a random subset of an index 772

1.MaxCompute

1.1. Java SDK

1.1.1. SDK for Java

This topic describes the most commonly used APIs of MaxCompute SDK for Java. For more information, visit the following website:

Comprehensive APIs of MaxCompute

You can use Maven to configure the new version of the SDK. The following configuration is used:

```
<dependency>
 <groupId>com.aliyun.odps</groupId>
 <artifactId>odps-sdk-core</artifactId>
 <version>xxxx-public</version>
</dependency>
```

The following table describes the overall information about SDK packages provided by MaxCompute.

Package	Description
odps-sdk-core	Basic features of MaxCompute. This package encapsulates MaxCompute APIs such as odps, and basic features such as projects and tables. It also includes Tunnel-related features.
odps-sdk-core-internal	Extended features of MaxCompute. This package encapsulates the less commonly used MaxCompute features and operations, such as Event and XFlow.
odps-sdk-commons	MaxCompute infrastructure. This package includes TableSchema, Column, Record, OdpsType, and some util packages.
odps-sdk-udf	The UDF API for MaxCompute.
odps-sdk-mapred	The MapReduce API for MaxCompute.
odps-sdk-graph	The Graph API for MaxCompute.

The SDK packages are available at SDK package download.

> **Note** If your MaxCompute version is earlier than V3.8.0, download the **0.27.2-public/** version of SDK packages. If your MaxCompute version is V3.8.0 or later, download the **0.30.8-public/** or **0.30.9-public/** version of SDK packages.

1.1.2. Prerequisites

Before using the SDK for Java, you must perform the following operations:

- Create an account with authorization to access MaxCompute and obtain the AccessKey pair for the account. For more information about how to obtain and view your AccessKey pair in the Apsara Stack console, see Obtain an AccessKey pair.
- Obtain the endpoint. For more information, see Obtain an endpoint.

1.1.2.1. Overview

Before using SDKs for MaxCompute, you must perform the following operations:

- Create an account that is authorized to access MaxCompute and obtain an AccessKey pair (AccessKey ID and AccessKey secret).
- Obtain the service endpoint.

1.1.2.2. Obtain an AccessKey pair

You can obtain an AccessKey pair in the Apsara Stack Cloud Management (ASCM) console.

1.1.2.2.1. 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.

1.1.2.2.2. Obtain the AccessKey pair of an organization

This topic describes how to obtain an AccessKey pair.

Prerequisites

Only the operation administrators or level-1 organization administrators can obtain the AccessKey pair of an organization.

Procedure

1. Log on to the Apsara Stack Cloud Management (ASCM) console as an administrator.
2. In the top navigation bar, click **Enterprise**.
3. In the left-side navigation pane of the **Enterprise** page, click **Organizations**.
4. In the Organizations navigation tree, click the icon on the right of an organization.
5. Select **AccessKey** from the shortcut menu.
6. In the message that appears, view the AccessKey pair of the organization.

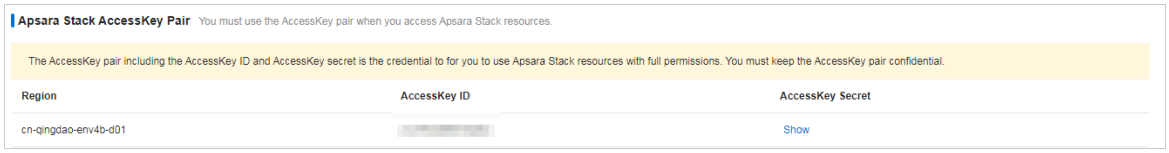
> **Note** An AccessKey pair is automatically allocated to a level-1 organization. Subordinate organizations of the level-1 organization use the AccessKey pair of the level-1 organization.

1.1.2.2.3. View the AccessKey pair of your Apsara Stack tenant account

To secure cloud resources, the system must verify the identity of visitors and ensure that they have the relevant permissions. You must obtain the AccessKey ID and AccessKey secret of your personal account to access cloud resources.

Procedure

1. Log on to the Apsara Stack Cloud Management (ASCM) console as an administrator.
2. In the upper-right corner of the homepage, move the pointer over the profile picture and click **User Information**.
3. In the **Apsara Stack AccessKey Pair** section of the User Information page, view your AccessKey pair.



> **Note** The AccessKey pair is made up of the AccessKey ID and AccessKey secret. These credentials provide you full permissions on Apsara Stack resources. You must keep the AccessKey pair confidential.

1.1.2.3. Obtain the endpoint

This topic describes how to obtain the service endpoint of DataHub in the Apsara Stack Operations (ASO) console.

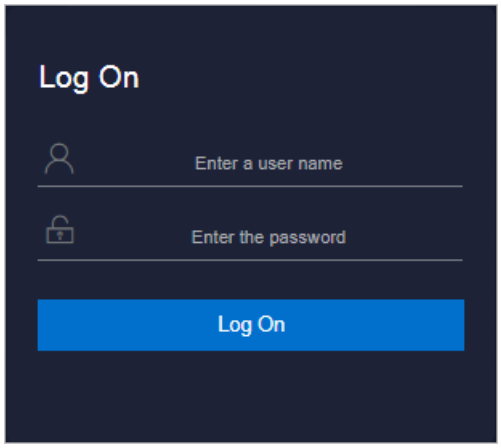
Context

- The URL, username, and password that are required to log on to the ASO console are obtained from deployment engineers or administrators.

  The URL of the ASO console is in the following format: *region-id*.aso.*intranet-domain-id*.com.
- A browser is available. We recommend that you use Google Chrome.

Procedure

1. Open your browser.
2. In the address bar, enter the URL *region-id*.aso.*intranet-domain-id*.com and press Enter.

   > **Note** You can select a language from the drop-down list in the upper-right corner of the page.

3. Enter your username and password.

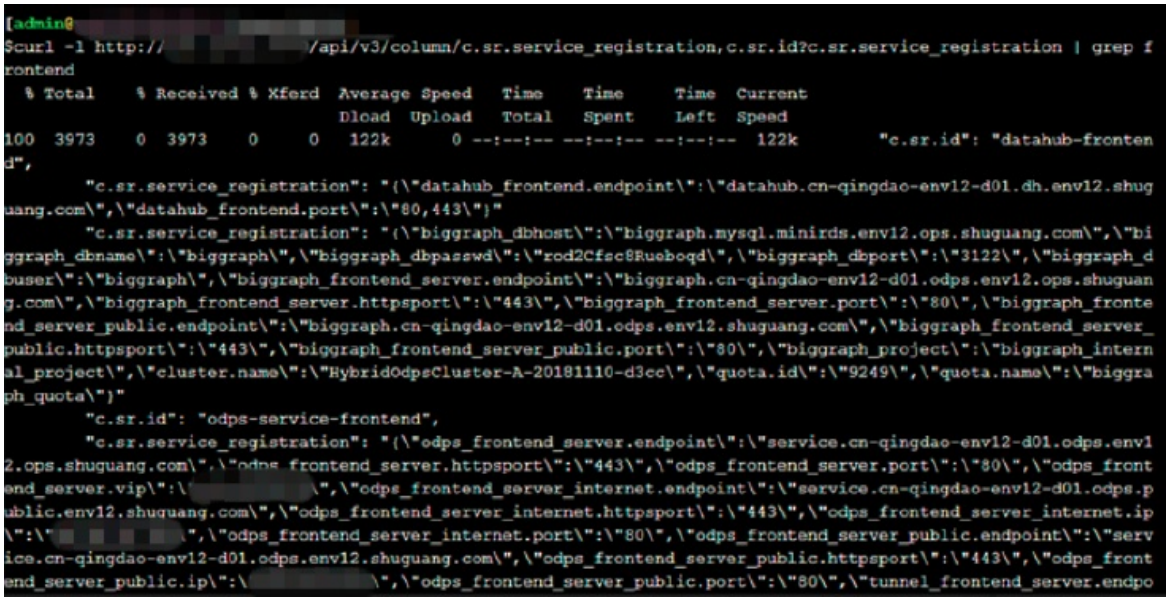
   > **Note** You can obtain the username and password for logging on to the ASO console from deployment engineers or administrators.

   - By default, the following user roles are available:
     - Security administrator: the user who has the permissions to manage other users or roles.
     - Auditor: the user who has the permissions to view audit logs.
     - Sysadmin: the user who has more permissions than a security administrator and an auditor.
   - When you log on to the ASO console for the first time, you must change the password as prompted.

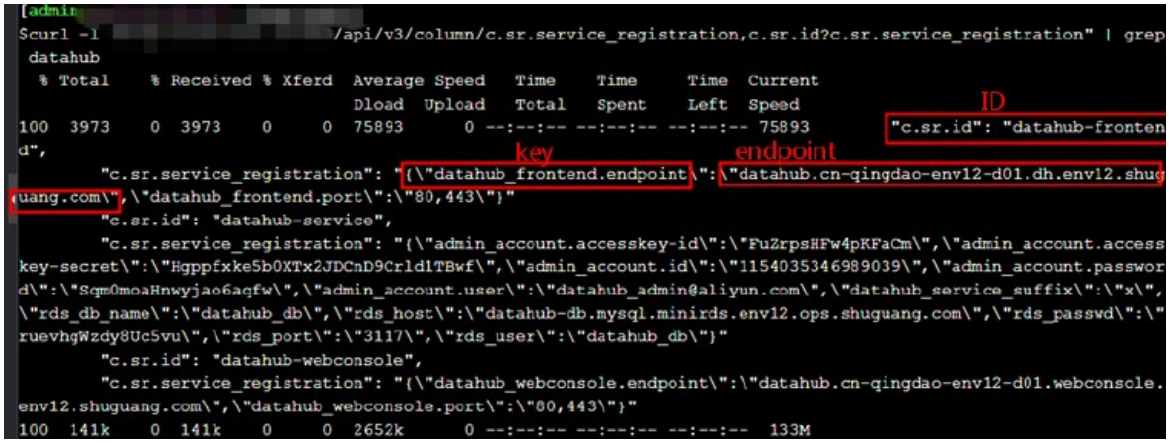
     To enhance security, make sure that the password meet the following requirements:
     - The password contains uppercase and lowercase letters.
     - The password contains digits.
     - The password contains special characters such as exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%).

- The password is 10 to 20 characters in length.

4. Click **Log On** to go to the **ASO** console.
5. In the left-side navigation pane, choose **Products > Product List**. In the **Basic O&M** section of the **Apsara Stack O&M** pane, click **Apsara Infrastructure Management Framework**.
6. In the left-side navigation pane, choose **Operations > Machine Operations**. The **Machine Operations** page appears.
7. In the Machines search box, search for a host. Click **Terminal** in the Actions column of the host to log on to the host.
8. Run the following command on the host to obtain JSON data: **curl -l "http://127.0.0.1:7070/api/v3/column/c.sr.service_registration,c.sr.id?c.sr.service_registration"**.



9. Find the endpoint of *DataHub* based on the service ID (*datahub-frontend*) and the key value (*datahub_frontend.endpoint*) of *DataHub*.



1.1.3. Major APIs

1.1.3.1. AliyunAccount

The primary account created with Alibaba Cloud. The input parameters are AccessKey ID and AccessKey Secret. AccessKey ID uniquely identifies an Alibaba Cloud user, and AccessKey Secret is used to authenticate the identity of the user. This class is used to initialize MaxCompute.

1.1.3.2. Odps

It is the entrance to MaxCompute SDK. You can use this class to obtain all object collections in a project, including Projects, Tables, Resources, Functions, and Instances.

You can pass this class into the AliyunAccount instance to construct a MaxCompute object. Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Table t : odps.tables()) {
....
}
```

> **Note** MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.3. Projects

A collection of all projects in MaxCompute.

A project is an element in a collection. Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps  odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Project p = odps.projects().get("my_exists");
p.reload();
Map<String, String> properties = prj.getProperties();
...
```

> **Note** MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.4. Project

A description of project information. You can obtain a list of projects through Projects.

1.1.3.5. SQLTask

An API for running and processing SQL tasks. You can use SQLTask.run to run SQL tasks. The SQLTask.run API returns an instance, and obtains the running status and results of SQL from the instance.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key"); Odps odps = new Odps(account)
;
String  odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Instance instance = SQLTask.run(odps, "my_project", "select ...") ;
String id = instance.getId();
instance.waitforsuccess();
Set<String> taskNames = instance.getTaskNames();
for (String name
taskNames) {
TaskSummary summary = instance.getTaskSummary(name);
String s = summary.getSummaryText();
}
Map<String, String> results = instance.getTaskResults();
Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
String result = results.get(status.getKey());
}
```

> **Note** To create a table, you must use the SQLTask API instead of the Table API. You must pass the CREATE TABLE statement into SQLTask. MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.6. Instances

A collection of all instances in MaxCompute.

An instance is an element in a collection. Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Instance i : odps.instances ()) {
....
}
```

Note MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.7. Instance

A description of instance information. You can obtain a list of instances through Instances.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps (account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Instance ins = odps.instances().get("instance id");
Date startTime = instance.getStartTime();
Date endTime = instance.getEndTime();
...
Status instanceStatus = instance.getStatus();
String instanceStatusStr = null;
if (instanceStatus == Status.TERMINATED) {
instanceStatusStr = TaskStatus.Status.SUCCESS.toString();
Map<String, TaskStatus> taskStatus = instance.getTaskStatus();
for (Entry<String, TaskStatus> status : taskStatus.entrySet()) {
if (status.getValue().getStatus() ! = TaskStatus.Status.SUCCESS) {
instanceStatusStr = status.getValue().getStatus().toString();
break;
}
}
} else {
instanceStatusStr = instanceStatus.toString();
}
...
TaskSummary summary = instance.getTaskSummary("instance name");
String s = summary.getSummaryText();
```

Note MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.8. Tables

A collection of all tables in MaxCompute. A table is an element in a collection.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Table t : odps.tables()) {
....
}
```

Note MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.9. Table

A description of table information. You can obtain a list of tables through Tables.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Table t = odps.tables().get("table name");
t.reload();
Partition part = t.getPartition(new PartitionSpec(tableSpec[1]));
part.reload();
...
```

Note MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.10. Resources

A collection of all resources in MaxCompute. A resource is an element in a collection.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Resource r : odps.resources()) {
....
}
```

> **Note** MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.11. Resource

A description of resource information. You can obtain a list of resources through Resources.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Resource r = odps.resources().get("resource name");
r.reload();
if (r.getType() == Resource.Type.TABLE) { TableResource tr = new TableResource(r);
String tableSource = tr.getSourceTable().getProject() + "." + tr.getSourceTable().getName();
if (tr.getSourceTablePartition() ! = null) {
tableSource += " partition(" + tr.getSourceTablePartition().toString() + ")";
}
....
}
```

> **Note** MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

The following sample code shows how to create a file resource:

```
String projectName = "my_porject";
String source = "my_local_file.txt";
File file = new File(source);
InputStream is = new FileInputStream(file);
FileResource resource = new FileResource();
String name = file.getName();
resource.setName(name);
odps.resources().create(projectName, resource, is);
```

The following sample code shows how to create a table resource:

```
TableResource resource = new TableResource(tableName, tablePrj, partitionSpec);
//resource.setName(INVALID_USER_TABLE);
resource.setName("table_resource_name");
odps.resources().update(projectName, resource);
```

1.1.3.12. Functions

A collection of all functions in MaxCompute. A function is an element in a collection.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
odps.setDefaultProject("my_project");
for (Function f : odps.functions()) {
....
}
```

> **Note** MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

1.1.3.13. Function

A description of function information. You can obtain a list of functions through Functions.

Sample code:

```
Account account = new AliyunAccount("my_access_id", "my_access_key");
Odps odps = new Odps(account);
String odpsUrl = "<your odps endpoint>";
odps.setEndpoint(odpsUrl);
Function f = odps.functions().get("function name");
List<Resource> resources = f.getResources();
```

> **Note** MaxCompute provides two endpoints. For more information, see *Tunnel SDK overview*.

The following sample code shows how to create a function:

```
String resources = "xxx:xxx";
String classType = "com.aliyun.odps.mapred.open.example.WordCount";
ArrayList<String> resourceList = new ArrayList<String>();
for (String r : resources.split(":")) {
resourceList.add(r);
}
Function func = new Function();
func.setName(name);
func.setClassType(classType);
func.setResources(resourceList);
odps.functions().create(projectName, func);
```

1.1.3.14. Spark Shell

An API commonly used by Spark to submit tasks to a cluster.

Run the following commands to enable the API:

```
$cd $SPARK_HOME
-- Access the spark directory.
$bin/spark-shell --master yarn
-- Select the running mode and enable the API.
```

Sample code:

```
sc.parallelize(0 to 100, 2).collect
sql("show tables").show
sql("select * from spark_user_data").show(200,100)
```

1.1.3.15. Spark R

An API commonly used by Spark to submit tasks to a cluster.

Run the following commands to enable the API:

```
$mkdir -p /home/admin/R && unzip . /R/R.zip -d /home/admin/R/
-- Create a directory named R and decompress R.zip in the directory.
$export PATH=/home/admin/R/bin/:$PATH
-- Configure environment variables.
$bin/sparkR --master yarn --archives . /R/R.zip
-- Select a running mode and enable the API.
```

Sample code:

```
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)
```

1.1.3.16. Spark SQL

An API commonly used by Spark to submit tasks to a cluster.

Run the following commands to enable the API:

```
$cd $SPARK_HOME
-- Access the spark directory.
$bin/spark-sql --master yarn
-- Select the running mode and enable the API.
```

Sample code:

```
show tables;
select * from spark_user_data limit 3;
quit;
```

1.1.3.17. Spark JDBC

An API commonly used by Spark to submit tasks to a cluster.

Run the following commands to enable the API:

```
$sbin/stop-thriftserver.sh
-- Stop a thread.
$sbin/start-thriftserver.sh
-- Restart a thread.
$bin/beeline
-- Enable the API.
```

Sample code:

```
! connect jdbc:hive2://localhost:10000/odps_smoke_test
show tables;
select * from mr_input limit 3;
! quit
```

1.1.3.18. Document APIs

An open-source API repository compatible with Elasticsearch on MaxCompute. For more information about the APIs, see the following link:

Document APIs

1.1.3.19. Search APIs

An open-source API repository compatible with Elasticsearch on MaxCompute. For more information about the APIs, see the following link:

Search APIs

1.2. SDK for Python

This topic describes how to use SDK for Python to create, view, and delete tables.

PyODPS is MaxCompute SDK for Python. PyODPS provides the DataFrame framework and basic operations on MaxCompute objects to allow you to easily analyze data in MaxCompute. For more information, visit GitHub and Docs that include details about all interfaces and classes.

For more information about PyODPS, see *MaxCompute User Guide*.

The following sections describe how to use SDK for Python to create, view, and delete tables.

> **Note** The following examples are for reference only.

Install PyODPS

PyODPS supports Python 2.6 and later (including Python 3). After you install pip in the system, you only need to run the `pip install pyodps` command. The related dependencies of PyODPS are automatically installed. For more information, see the **Installation guide** section in *MaxCompute User Guide*.

Perform initialization

Use your Apsara Stack tenant account to initialize a MaxCompute entry object. Example:

```
from odps import ODPS
odps = ODPS('**your-access-id**', '**your-secret-access-key**', '**your-default-project**',
        endpoint='**your-end-point**')
```

After initialization, you can perform operations on tables, resources, and functions.

> **Note** Unless otherwise specified, objects in the following sections all refer to MaxCompute entry objects.

Obtain a PyODPS project

A project is the basic organizational unit in MaxCompute, similar to a database.

You can use the `get_project` method to obtain information about a project. Example:

```
project = odps.get_project('my_project')  # Obtain information about a project.
project = odps.get_project()              # Obtain information about the default project.
```

> **Note**
>
> - If no parameters are specified, information about the default project is obtained.
> - You can use the exist_project method to check whether a project exists.
> - A table is the data storage unit in MaxCompute.

Perform operations on PyODPS tables

Call `list_tables` to query all tables in a project. Example:

```
for table in odps.list_tables():  # Process each table.
```

Call `exist_table` to check whether a table exists. Call `get_table` to obtain information about a table.

```
t = odps.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.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>]
```

Create a schema for a PyODPS table

Use one of the following methods to initialize the schema of a PyODPS table:

- Initialize the schema based on table columns and optional partitions.

```
from odps.models import Schema, Column, Partition
columns = [Column(name='num', type='bigint', comment='the column')]
partitions = [Partition(name='pt', type='string', comment='the partition')]
schema = Schema(columns=columns, partitions=partitions)
schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

- Call `Schema.from_lists`. Although this method is more convenient, you cannot directly set comments for columns and partitions.

```
schema = Schema.from_lists(['num'], ['bigint'], ['pt'], ['string'])
schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

Create a PyODPS table

You can create a table based on the table schema. Example:

```
table = odps.create_table('my_new_table', schema)
table = odps.create_table('my_new_table', schema, if_not_exists=True)  # Create a table only if no table wit
h the same name exists.
table = o.create_table('my_new_table', schema, lifecycle=7)  # Configure the lifecycle for a table.
```

You can also use a comma-connected string in the format of Field name, Field data type to create a table. Example:

```
# Create a non-partitioned table.
table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
# To create a partitioned table, you can specify (Table field list, Partition field list).
table = o.create_table('my_new_table', ('num bigint, num2 double', 'pt string'), if_not_exists=True)
```

By default, when you create a table, you can use only the following data types: BIGINT, DOUBLE, DECIMAL, STRING, DATETIME, BOOLEAN, MAP, and ARRAY.

If you want to use other data types, such as TINYINT and STRUCT, set `options.sql.use_odps2_extension` to True. Example:

```
from odps import options
options.sql.use_odps2_extension = True
table = o.create_table('my_new_table', 'cat smallint, content struct<title:varchar(100), body string>')
```

Obtain PyODPS table data

Use one of the following methods to obtain table data:

- Call the `head` method 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.
```

- Call the `open_reader` method for a table 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 the statement 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]
    # Process a record.
```

- Call the Tunnel API to read table data. The `open_reader` method is an encapsulation of the Tunnel API.

Use PyODPS to write data

Similar to `open_reader`, you can use `open_writer` for a table 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:
    writer.write(records) # records can be iterable records and are written to block 0 by default.
with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer: # Enable two blocks.
    writer.write(0, gen_records(block=0))
    writer.write(1, gen_records(block=1)) # The two write operations can be performed in parallel by using th
e multi-threading technology. 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.
```

Writing data to a table is an encapsulation of the Tunnel API.

Delete a PyODPS table

Delete a table. Example:

```
odps.delete_table('my_table_name', if_exists=True) # Delete a table only if the table exists.
t.drop() # Directly drop a table if the table exists.
```

Perform operations on PyODPS table partitions

- Perform basic operations

  Traverse all partitions of a table. Example:

  ```
  for partition in table.partitions:
      print(partition.name)
  for partition in table.iterate_partitions(spec='pt=test'):
      # Traverse subpartitions.
  ```

  Check whether a partition exists. Example:

  ```
  table.exist_partition('pt=test,sub=2019')
  ```

  Obtain information about a partition. Example:

  ```
  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 a partition only if no partition with the same na
  me exists.
  ```

- Delete a partition

  ```
  t.delete_partition('pt=test', if_exists=True) # Delete a partition only if the partition exists.
  partition.drop() # Directly drop a partition if the partition exists.
  ```

PyODPS SQL

PyODPS supports MaxCompute SQL queries and provides methods to obtain execution results.

- Run SQL statements

```
odps.execute_sql('select * from dual')  # Execute the SQL statement in synchronous mode. Blocking continues until the execution of the SQL statement is complete.
instance = odps.run_sql('select * from dual')  # Execute the SQL statement in asynchronous mode.
instance.wait_for_success()  # Blocking continues until the execution of the SQL statement is complete.
```

- 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 odps.execute_sql('select * from dual').open_reader() as reader:
    for record in reader:
      # Process each record.
  ```

  If the `desc` command is run, you can use the `reader.raw` method to obtain the raw SQL query results.

  ```
  with odps.execute_sql('desc dual').open_reader() as reader:
    print(reader.raw)
  ```

PyODPS resources

Resources commonly apply to UDFs and MapReduce on MaxCompute.

You can call the `list_resources` method to query all resources, the `exist_resource` method to check whether a resource exists, and the `delete_resource` method to delete a resource. You can also call the `drop` method to delete a resource.

PyODPS supports two resource types: files and tables.

- File resources

  Files of basic file types and the .py, jar, and .archive types are supported.

  - Create a file resource

    You can specify a resource name, a file type, and a file-like or string object to create a file resource. 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.
    ```

- Read and modify a file resource

  You can call the `open` method for a file resource or call the `open_resource` method at the MaxCompute entry to open a file resource. The opened object is a file-like object. Similar to the `open` method built in Python, file resources also support various opening modes. 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 to 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 to the file.
  fp.write('Hello World')
  fp.flush()  # Manually calling the method to submit the update to MaxCompute.
```

  PyODPS supports the following opening modes:

  - r: read mode. The file can be opened but data cannot be written to it.
  - w: write mode. Data can be written to the file, but data in the file cannot be read. If a file is opened in write mode, the file content is cleared first.
  - a: append mode. Data can be added to the end of the file.
  - r+: read/write mode. You can read data from and write data to the file.
  - w+: This mode is similar to 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

```
odps.create_resource('test_table_resource', 'table', table_name='my_table', partition='pt=test')
```

  Update a table resource

```
table_resource = odps.get_resource('test_table_resource')
table_resource.update(partition='pt=test2', project_name='my_project2')
```

DataFrame

PyODPS provides a pandas-like API called PyODPS DataFrame. This API makes full use of the computing power of MaxCompute. For more information, see DataFrame.

The following DataFrame examples are provided for reference only.

> **Note** You must create a MaxCompute object before you perform subsequent steps.
>
> ```
> odps = ODPS('**your-access-id**', '**your-secret-access-key**',
>         project='**your-project**', endpoint='**your-end-point**'))
> ```

Assume that the following tables exist: pyodps_ml_100k_movies that contains movie-related data, pyodps_ml_100k_users that contains user-related data, and pyodps_ml_100k_ratings that contains rating-related data.

You only need to specify a table object to create a DataFrame object. Example:

```
from odps.df import DataFrame
```

```
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

You can use the dtypes attribute to view the fields of DataFrame and the types of these fields. Example:

```
users.dtypes
```

You can use the `head` method to obtain the first N data records for a 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

	user_id	age	sex	occupation	zip_code
9	10	53	M	lawyer	90703

You can add a filter condition on the fields to view the required fields. 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

You can also exclude several fields from results. 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

If you exclude some fields, you may want to obtain new columns by using 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

	user_id	age	occupation	sex_bool
3	4	24	technician	True
4	5	33	other	False

Obtain the number of users aged between 20 and 25. Example:

```
users.age.between(20, 25).count().rename('count')
943
```

Obtain the numbers of male and female users. Example:

```
users.groupby(users.sex).count()
```

	sex	count
0	F	273
1	M	670

To divide users by occupation, obtain the top 10 occupations, and sort the occupations in descending order based on population. Example:

```
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

The DataFrame API provides the value_counts method for the same purpose. Example:

```
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

Show data in an intuitive graph. Example:

```
%matplotlib inline
```

Use a horizontal column chart to visualize data. Example:

```
users['occupation'].value_counts().plot(kind='barh', x='occupation',
ylabel='prefession')
```

Divide users into 30 groups by age and view the histogram of age distribution. Example:

```
users.age.hist(bins=30, title="Distribution of users' ages", xlabel='age', ylabel='count of users')
```

Use the join method to join three tables and save the joined tables as a new table. Example:

```
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
}
```

Divide users aged 0 to 80 into eight age groups. Example:

```
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')]
```

View the first 10 data records of a single age in a group. Example:

```
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

View the total rating and average rating of users in each age group. Example:

```
cut_lens.groupby('age_group').agg(cut_lens.rating.count().rename('total_rating'), cut_lens.rating.mean().rename('average_rating'))
```

	age_group	average_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

Configuration

PyODPS provides a series of configuration options, which can be obtained by using `odps.options`. The following table describes configurable MaxCompute options.

- General configurations

Option	Description	Default
end_point	MaxCompute endpoint.	N/A
default_project	The default project.	N/A
log_view_host	The hostname of Logview.	N/A
log_view_hours	The retention time of Logview. Unit: hours.	24
local_timezone	The time zone. True indicates local time, and False indicates UTC. The pytz time zone can also be used.	1
lifecycle	The lifecycle of all tables.	N/A
temp_lifecycle	The lifecycle of temporary tables.	1
biz_id	The ID of the user.	N/A

Option	Description	Default
verbose	Specifies whether to display logs.	False
verbose_log	The log receiver.	N/A
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 time to wait before the connection times out.	5
read_timeout	The time to wait before the read operation times out.	120
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.	N/A
sql.use_odps2_extension	Specifies whether to enable MaxCompute V2.0 language extension.	False

- Data upload/download configurations

Option	Description	Default
tunnel.endpoint	Tunnel endpoint.	N/A
tunnel.use_instance_tunnel	Specifies whether to use Instance Tunnel to obtain execution results.	True
tunnel.limited_instance_tunnel	Specifies whether to limit the number of data records obtained by using Instance Tunnel.	True

Option	Description	Default
tunnel.string_as_binary	Specifies whether to use bytes instead of unicode for data of the STRING type.	False

- DataFrame configurations

Option	Description	Default
interactive	Specifies whether DataFrame is used in an interactive environment.	Depending on the measured 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 a pair of grave accents (``) to mark fields and table names in the backend of MaxCompute SQL.	True
df.libraries	The resource name of the third-party library that is used for DataFrame operations.	N/A

- PyODPS ML configurations

Option	Description	Default
ml.xflow_project	The default XFlow project name.	algo_public
ml.use_model_transfer	Specifies whether to use ModelTransfer to obtain the model PMML.	True
ml.model_volume	The volume name used when ModelTransfer is used.	pyodps_volume

2.DataWorks

2.1. Overview

DataWorks is a one-stop big data analytics and governance service released by Alibaba Cloud in 2009. It integrates all processes from data collection to data display and from data analytics to application running. It offers a wide range of features and services to help enterprises effectively build a data mid-end.

2.2. Terms

Workspace

A workspace is the basic organizational object in DataWorks. It serves as the basic unit for you to manage tables, resources, user-defined functions (UDFs), nodes, and permissions.

Node

A node is a scheduling definition that is submitted and released on DataStudio or created by calling APIs.

Business flow

A business flow consists of a set of nodes with dependency between each other.

Instance or task instance

Before nodes can be run, they must be converted into task instances in the DataWorks scheduling system.

2.3. List of operations by function

Operations of the scheduling system

- Create a node
- Update a node
- Update multiple nodes at a time
- Delete a node
- Query parent nodes
- Query child nodes
- Query the node code
- Query nodes
- Query a node by ID
- Query the total number of nodes
- Query node instances
- Query the operational logs of a node instance
- Query a node instance by ID
- Query the parent node at a specified level

- Query the child node instances at a specified level
- Query the number of node instances in each status of an application
- Query the number of task instances in each status on a specified date
- Query the number of node instances in each status for applications
- Rerun an instance
- Restore an instance
- Set the instance status to Success and trigger descendant instances
- Terminate a node
- Rerun a descendant instance
- Update the scheduling configurations of an instance
- Pause an instance
- Terminate multiple instances at a time
- Rerun multiple instances at a time
- Restore multiple nodes at a time
- Pause multiple instances at a time
- Set the status of multiple instances to Success
- Query the total number of instances on the current day
- Query workflow instances
- Create a retroactive data generation task
- Run a smoke test
- Run an ad-hoc workflow
- Terminate a workflow
- Submit a resource package
- View the submission status of the resource package

2.4. Make API requests

DataWorks allows you to make API requests by using the Java SDK or the HTTP method. The Java SDK is newly released. You can use only the Java SDK to call API operations in the later versions of DataWorks. The HTTP method remains unchanged. No new API operations are added. In later versions, these API operations will be discontinued. We recommend that you use the Java SDK.

2.4.1. Java SDK

If you need to use a client to make an API request by using the Java SDK, you must specify the following parameters to initialize the client: popEndpoint, accessId, accessKey, popProduct, regionId, and popEndpointName.

popEndpoint

The endpoint for accessing the service. It consists of the prefix dataworks. followed by the domain name of Apsara Stack DataWorks. For example, if the DataStudio URL for Apsara Stack is http://ide.envxxx.yyyyy.com, the endpoint is dataworks. envxxx.yyyyy.com. If the actual endpoint does not follow the preceding rule, contact Alibaba Cloud technical support.

accessId and accessKey

The AccessKey ID and AccessKey secret of the Apsara Stack tenant account.

popProduct

The name of the service. Set the value to dataworks-private-cloud.

regionId and popEndpointName

- regionId: the ID of the region where the service resides. Set the value to default.
- popEndpointName: the name of the endpoint. Set the value to default.

Authorization

You can use only an authorized Apsara Stack tenant account to make API requests by using the Java SDK.

The whitelist mechanism is used for the access control of the DataWorks API. To authorize an Apsara Stack tenant account, add the account to the whitelist. If the account that you use to make API requests is not in the whitelist, the following error message appears.

```
com.aliyuncs.exceptions.ClientException: InvalidApi.NotFound : Specified api is not found, pleasecheck your url and method.
```

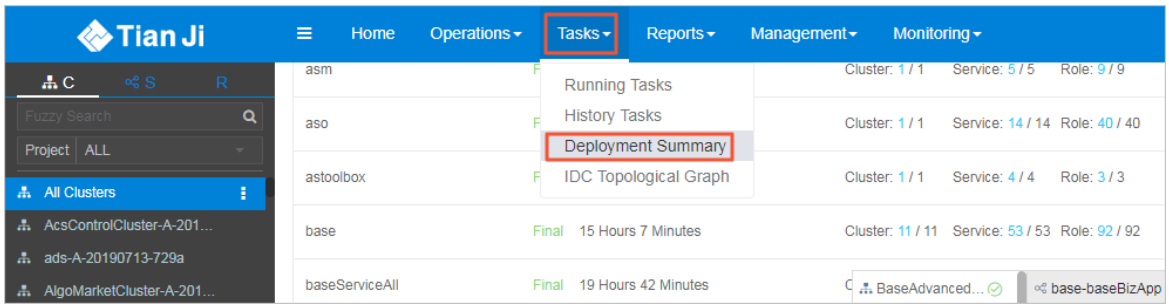
This error message indicates that the account is not authorized to call the specified API operation. To resolve this issue, perform the following operations:

Contact the DataWorks technical support staff and provide them with the Apsara Stack tenant account to be authorized and the API operations to be called. After you receive an encrypted license generated by the DataWorks technical support staff based on the configuration in the runtime environment, use the license to authorize the account to make API requests.

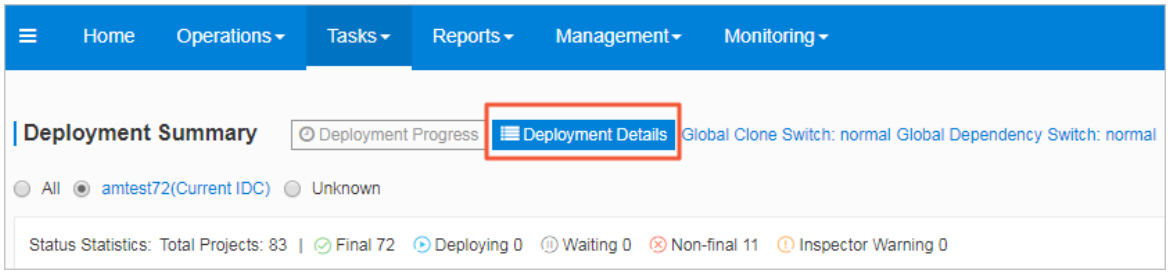
1. Log on to Apsara Infrastructure Management Framework.

   For more information, see *Log on to Apsara Infrastructure Management Framework* under **Operations of basic platforms > Apsara Infrastructure Management Framework** in *Alibaba Cloud Apsara Stack Enterprise Operations Guide*.

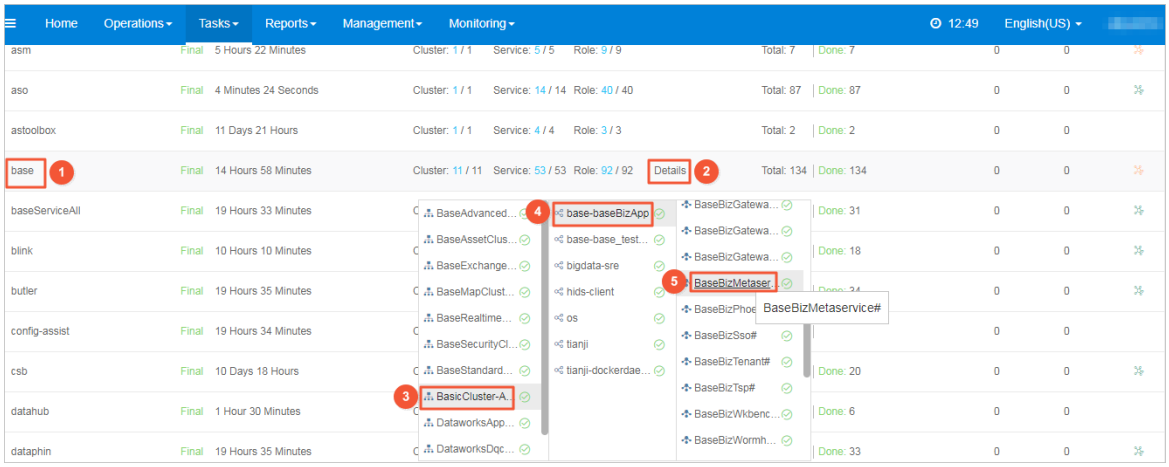
2. In the left-side navigation pane, click **Monitoring**.
3. In the top navigation bar, move the pointer over **Tasks** and select **Deployment Summary** from the drop-down menu.



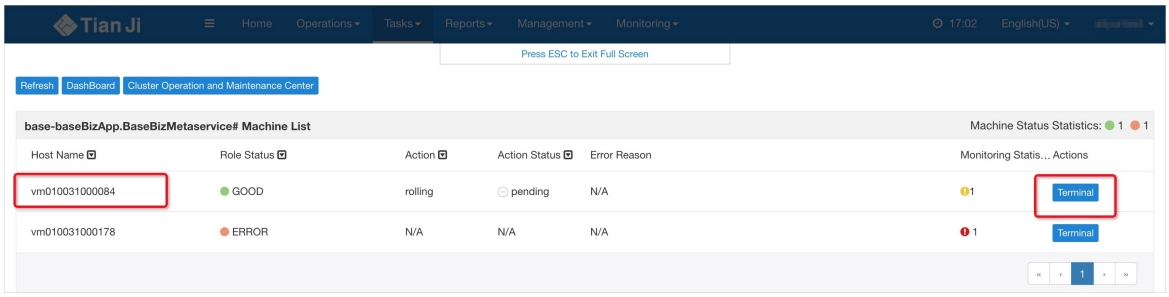
4. On the **Deployment Summary** page, click **Deployment Details**. The Deployment Details page appears.



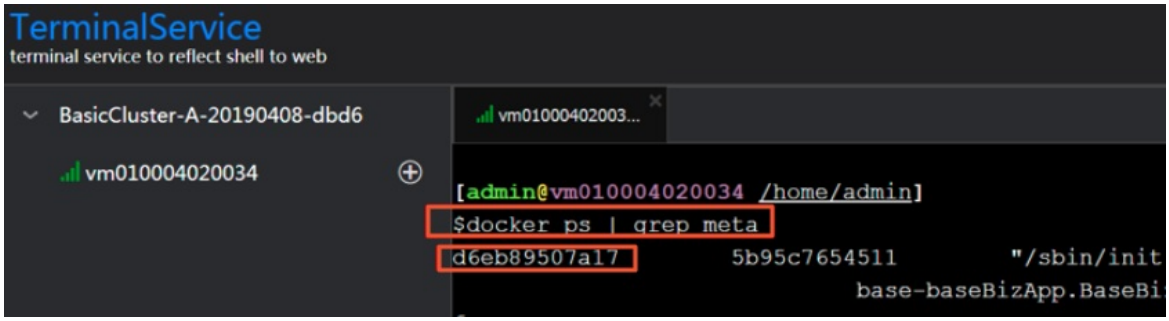
5. Move the pointer over the **base** service, click **Details**, and then choose **BasicCluster-A-20190708-376d Machine List > base-baseBizApp > BaseBizMetaservice**.



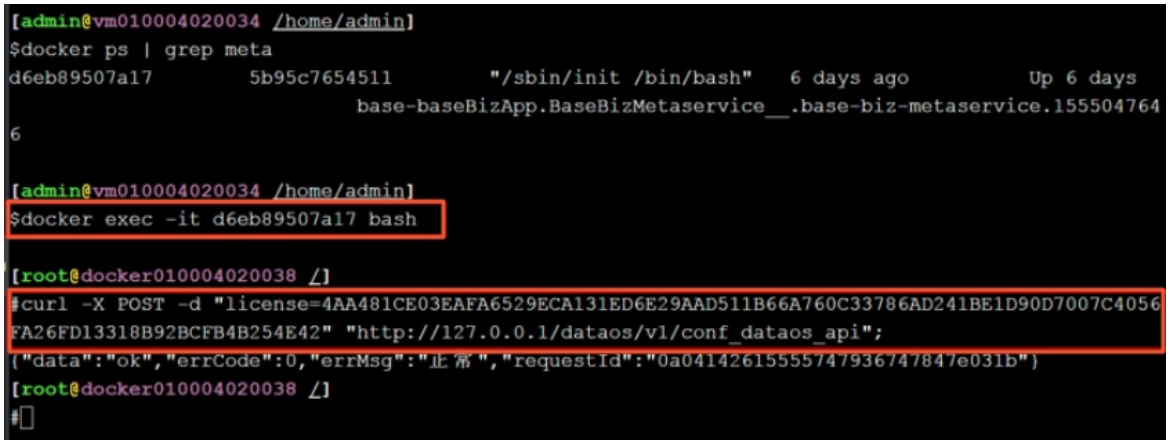
6. On the page that appears, find the target host and click **Terminal** in the Actions column to go to the TerminalService page.



7. On the TerminalService page, click the Add icon, and run the `docker ps | grep meta` command on the tab that appears to obtain the ID of the base-biz-metaservice container. For example, the following figure shows that the container ID is d6eb89507a17.



8. Log on to the container and run the docker exec -it d6eb89507a17 bash command. Then, you can use the license for authorization.



Run the following command to authorize the account to make API requests:

```
curl -X POST -d "license=4AA481CE03EAFA6529ECA131ED6E29AAD511B66A760C33786AD241BE1D90D700
7C4056FA26FD13318B92BCFB4B254E42" "http://127.0.0.1/dataos/v1/conf_dataos_api";
```

In this command, the license is provided by the DataWorks technical support staff. If `{"data":"ok","errCode":0,"errMsg":"success","requestId":"0a041426155557479367478 47e031b"}` appears after you run the command, you can use the authorized account to make API requests.

Sample code

```
public void setup () throws IOException, ClientException {
  popEndpoint = "dataworks. envxxx.yyyyy.com";
  popEndpointName = "default";
  accessId = "accessId";
  accessKey = "accessKey";
  regionId = "default";
  popProduct = "dataworks-private-cloud";
  X509TrustAll.ignoreSSLCertificate();
  DefaultProfile.addEndpoint(popEndpointName, regionId, popProduct, popEndpoint);
  IClientProfile profile = DefaultProfile.getProfile(regionId, accessId, accessKey);
  IAcsClient client = new DefaultAcsClient(profile);
  }
```

```
/** * Example: DemoApiRequest *
@throws ServerException *
@throws ClientException */
@Test
public void test_DemoApi() throws ServerException, ClientException {
  DemoApiRequest request = new DemoApiRequest();
  request.setKeyword("abc");
  DemoApiResponse response = client.getAcsResponse(request);
  System.out.println(gson.toJson(response));
  }
```

Specify the account for calling API operations that involve the scheduling system

If you call an API operation that involves the scheduling system, you must specify the following three common request parameters: accountId, accountPwd, and clientName. The following procedure describes how to specify these three parameters:

1. You can ask a field engineer to log on to the database that hosts the base-biz-tsp service. The name of the database is dwtsp.
2. The field engineer inserts a record into the cloudapi_account table to specify the accountId, accountPwd, and clientName parameters. These three parameters uniquely identify the account that is used for calling the API operation.

2.4.2. HTTP API

Warning Alibaba Cloud has stopped maintaining the DataWorks HTTP API. To be compatible with earlier versions, the HTTP method can still be used, but will be discontinued in later versions. Change to use the Java SDK as soon as possible.

2.4.2.1. Endpoints

The DataWorks API operations are categorized based on functions. Each type of API operations is accessed by using a specific domain name. For more information about the domain names, see the description about each type of API operations.

2.4.2.2. Protocol

All APIs use the HTTP protocol in Apsara Stack.

2.4.2.3. Request method

You can use the HTTP GET or POST request method to fulfill your intentions based on the description in the related API reference.

- If you use the GET method, an API operation returns the result based on the parameters in the query string.

- If you use the POST method, an API operation returns the result based on the parameters in the query string. The parameters in the request body are ignored.

2.4.2.4. Encoding

UTF-8 encoding is used in all the DataWorks APIs.

2.4.2.5. Signature method

Header

(Required) base_id

(Required) tenant_id

Signature

You must sign all API requests so that your identity can be verified. To sign an API request, follow these steps:

1. Add the following two parameters to the request URL: baseKey and timestamp.
2. Calculate a signature and add it to the HTTP header.
   - Sample GET request

     In this example, User B provides an HTTP GET API: /getapi. The name and age parameters are query parameters in the request URL. User A needs to use a URL that includes the following part to make the API request: `/getapi?name=fengyeng&age=20`. After User A obtains a token from the token center with the authentication passed such as AppCode, the baseKey and timestamp parameters are added to the request URL as query parameters. You can specify the authentication method to allow others to access the API operation. Therefore, the new URL includes the following part: `/getapi?name=fengyeng&age=20?baseKey=dsa134&timestamp=123456789`. A signature is then generated based on the token and query parameters in the new URL, and added to the request header by the requester as a field, for example, signature:dsafadfsa141fdsaf1.
   - Sample POST request

     In this example, User B provides an HTTP POST API: /postapi. User A needs to use a URL that includes the following part to make the API request: /postapi. After User A obtains a token from the token center, the baseKey and timestamp parameters are added to the request URL as query parameters. Therefore, the new URL includes the following part: /postapi?baseKey=dsa134&timestamp=123456789. A signature is then generated based on the token and query parameters in the new URL, and added to the request header by the requester as a field, for example, signature:dsafadfsa141fdsaf1.

2.4.2.6. Response parameters

```
{
  "requestId": <STRING>, # The request ID, which is used to locate the log entries for troubleshooting.
  "returnCode": "0", # A value of 0 indicates that the API operation is successful.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": <The returned result>
}
```

2.4.2.7. API example

Sample code

```
import org.apache.commons.lang3.StringUtils; import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient; import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpUriRequest; import org.apache.http.impl.client.HttpClientBuild
er;
import java.io.BufferedReader; import java.io.IOException; import java.io.InputStreamReader; import java.n
et.URLDecoder;
import java.security.MessageDigest; import java.util.Arrays;
public class SearchNodes {
public static void main(String[] args) throws Exception {
HttpClient httpClient = HttpClientBuilder.create().buil
d();
String uri = "http://baseapi. ***.com/v1.0/node/prod?exe cuteMethod=SEARCH&searchText=nodeTest";
HttpUriRequest request = createHttpRequest(
"base key",
"base token", uri
);
request.setHeader("tenant_id", "tenant ID"); request.setHeader("base_id", "Apsara Stack tenant account I
D");
HttpResponse response = httpClient.execute(request); BufferedReader bufferedReader = new BufferedRead
er(new
InputStreamReader(response.getEntity().getContent())); StringBuilder stringBuilder = new StringBuilder();
while(bufferedReader.ready()) {
String line = bufferedReader.readLine(); if (line == null) {
break;
}
stringBuilder.append(line);
}
System.out.println(stringBuilder.toString());
```

```
}
private static String byteArrayToHexString(byte[] byteArray) {
final char[] HEX_DIGITS = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b','c', 'd', 'e', 'f'};
String result = null;
int l = byteArray.length; char[] str = new char[l << 1]; int k = 0;
for (int i = 0; i < l; i++) {
byte byte0 = byteArray[i];
str[k++] = HEX_DIGITS[byte0 >>> 4 & 0xf];
str[k++] = HEX_DIGITS[byte0 & 0xf];
}
result = new String(str); return result;
}
public static String tokenSign(String param, String salt) t hrows Exception {
if (StringUtils.isEmpty(param) || StringUtils.isEmpty(salt)) {
 return null;
}
String query = URLDecoder.decode(param, "UTF-8");
String[] pStr = query.trim().split("&");
Arrays.sort(pStr);
StringBuilder buf = new StringBuilder(); for (int i = 0; i < pStr.length; ++i) {
buf.append(pStr[i]);
buf.append("&");
}
String paramStr = buf.substring(0, buf.length() - 1);
String sourceStr = salt + ":" + paramStr + ":" + salt;
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] byteArray = md.digest(sourceStr.getBytes("UTF-8"));
return byteArrayToHexString(byteArray);
}
protected static HttpUriRequest createHttpRequest(String baseKey, String baseToken, String uri) {
StringBuilder buf;
int index = uri.indexOf('?') ;
if (index == -1) {
  index = uri.length();
  buf = new StringBuilder(uri.length() + baseKey.length() + 13);
  buf.append(uri).append('?') ;
} else {
  buf = new StringBuilder(uri.length() + baseKey.length() + 13);
  buf.append(uri).append('&');
}
  buf.append("baseKey=").append(baseKey);
```

```
buf.append("&timestamp=").append(System.currentTimeMillis());
HttpUriRequest request = new HttpGet(buf.toString()); String signature;
try {
  signature = tokenSign(buf.substring(index + 1), baseToken);
} catch (Exception ex) {
 throw new RuntimeException("error on sign for uri: " + uri, ex);
}
request.addHeader("signature", signature);
return request;
}
}
```

2.5. APIs for HTTP requests

2.5.1. Create a node

URL: /v1.0/node/{projectEnv}/sync

Method: POST

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Body

```
{
  "executeMethod": CREATE_NODE_SYNC, # The type of the operation.
  # For more information about the parameters, see NodeModifyDto in Data structure.
  "nodeType": <INTEGER>, # The node type. Valid values: 0, 1, 2, and 3. The value 0 specifies a node that runs as scheduled. The value 1 specifies a one-off node. The value 2 specifies a paused node. The value 3 specifies a dry-run node.
  "dependentType": <INTEGER>, # The dependency type of the node. Valid values: 0, 1, 2, and 3. The value 0 specifies that the node can be run without waiting for the previous cycle of a specified node is completed. The value 1 specifies that the node is run only after the previous cycle of its child nodes is completed. The value 2 specifies that the node is run only after its previous cycle is completed. The value 3 specifies that the node is run only after the previous cycle of a custom node is completed.
  "nodeName": <STRING>, # The name of the node.
  "prgType": <INTEGER>, # The type of the node code.
  "priority": <INTEGER>, # The priority of the node.
  "owner": <STRING>, # The owner of the node.
  "appId": <LONG>, # The ID of the workspace to which the node belongs.
  "cycType": <INTEGER>, # The scheduling type.
  "codeSrc": <STRING>, # The node code.
  "nodeFrom": <STRING>, # The source of the node.
  "inputs": [{ # The input data of the node.
   "data": <STRING>, # The output data of the ancestor node.
  }],
  "outputs": [{ # The output data of the node.
    "data": <STRING>, # The output data of the current node.
  }],
"uniqueId": <STRING>, # The unique ID that you define for the request.
}
```

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is successful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "returnValue": <LONG>, # Node ID
}
```

2.5.2. Update a node

URL: /v1.0/node/{projectEnv}/sync

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": CREATE_NODE_SYNC, # The operation type.
  # For more information about the parameters, see NodeModifyDto in Data structure.
  "nodeType": <INTEGER>, # The node type. Valid values: 0, 1, 2, and 3. 0: a normal task. 1: an ad-hoc task. 2: a task of a paused node. 3: a task of a dry-run node.
  "dependentType": <INTEGER>, # The dependency type. Valid values: 0, 1, 2, and 3. 0:  The node is executed only after the current cycle of another node is completed. 1: The node is executed only after the previous cycle of the user-created node is completed. 2: The node is executed only after the previous cycle of the level-1 node is completed. 3: The node is executed only after its previous cycle is completed.
  "nodeName": <STRING>, # The name of the node.
  "prgType": <INTEGER>, # The type of the node script.
  "priority": <INTEGER>, # The priority.
  "owner": <STRING>, # The workspace owner.
  "appId": <LONG>, # The workspace ID.
  "cycType": <INTEGER>, # The scheduling type.
  "codeSrc": <STRING>, # The node code.
  "nodeFrom": <STRING>, # The source of the node.
  "inputs": [{ # The node inputs.
   "data": <STRING>,  # The output of the ancestor nodes.
  }],
  "outputs": [{ # The node outputs.
    "data": <STRING>, # The output of the node.
  }],
"uniqueId": <STRING>, # The unique ID that you define for the request.
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operatiaon details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.3. Update multiple nodes at a time

URL: /v1.0/node/{projectEnv}

Method: POST

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Body

```
[
  {
    "executeMethod": "CREATE_NODE", # The operation that you want to perform. Valid values: CREATE_NODE, UPDATE_NODE, and DELETE_NODE.
    "nodeId": <LONG>, # The ID of the node in the scheduling system. This parameter must be specified when you call this operation to update a node.
    "uniqueId": <LONG>, # Required. The unique ID of the request. It is transferred from the client and used only by the client to compare execution results.
    "nodeName": <STRING>, # Required. The name of the node.
    "nodeType": <INTEGER>, # Required. The node type. Valid values: 0, 1, 2, and 3. The value 0 specifies a node that runs as scheduled. The value 1 specifies a one-off node. The value 2 specifies a paused node. The value 3 specifies a dry-run node.
    "prgType": <INTEGER>, # Required. The type of the node code, for example, odps_shell and odps_sql.
    "description": <STRING>, # Optional. The description of the node.
    "paraValue": <STRING>, # Optional. The key-value pairs of a node.
    "priority": <INTEGER>, # Optional. The priority of the node. Default value: 0.
    "cronExpress": <STRING>, # Required. The CRON expression.
    "cloudUUID": <LONG>, # The node ID that is stored in the code library. This parameter is required when y
```

```
ou call this operation to update a node and is optional when you call this operation to create a node.
    "fileId": <LONG>, # Optional. The ID of the node code file.
    "fileVersion": <LONG>, # Optional. The version number of the node code file.
    "owner": <STRING>, # Required. The owner of the node. The value must be a standard employee ID.
    "resGroupId": <LONG>, # Required. The ID of the resource group to which the node belongs.
    "appId": <LONG>, # Required. The ID of the workspace to which the node belongs.
    "baseLineId": <LONG>, # The ID of the baseline to which the node belongs.
    "createUser": <STRING>, # Required. The user who created the node.
    "modifyUser": <STRING>, # Required. The user who modified the node last time.
    "multiInstCheckType": <INTEGER>, # Optional. The identifier of the operation that checks whether multi
ple instances exist.
    "cycType": <INTEGER>, # Required. The cycle type. Valid values: 0, 1, and 2. The value 0 specifies that the
cycle is equal to or longer than one day. The value 1 specifies that the cycle is shorter than one day. The valu
e 2 specifies that the cycle is shorter than one day and node instances are executed in order of the node inst
ance IDs, such as 1, 2, 3...
    "dependentType": <INTEGER>, # Required. The dependency type of the node. Valid values: 0, 1, 2, and 3.
The value 0 specifies that the node can be run without waiting for the previous cycle of a specified node is co
mpleted. The value 1 specifies that the node is run only after the previous cycle of its child nodes is complet
ed. The value 2 specifies that the node is run only after its previous cycle is completed. The value 3 specifies
that the node is run only after the previous cycle of a custom node is completed.
    "dependentNodeIds": [<LONG>, ...], # The ID of nodes on which the current node depends. By default, thi
s parameter is set to the IDs of child nodes of the current node.
    "multiInstKillType": <INTEGER>, # Optional. The type of instances to be terminated. Valid values: 0, 1, an
d 2. The value 0 specifies to terminate no node instances. The value 1 specifies to terminate non-routine nod
e instances. The value 2 specifies to terminate routine node instances.
    "refreshToTask": <BOOLEAN>, # Optional. Specifies whether the configuration takes effect on the curren
t node instance.
    "updateNodeOnly": <BOOLEAN>, # Specifies whether to refresh the metadata of ancestor and descenda
nt nodes. Default value: false.
    "codeSrc": <STRING>, # The node code. This parameter must be specified when the value of updateNode
Only is set to false.
    "envType": <INTEGER> , # Optional. The type of the runtime environment.
    "nodeFrom": <STRING>, # Optional. The source of the node, that is, the system from which the node is d
eployed.
    "inputs": [
      {
        "data": <STRING>, # Required. The input data of the node.
      },
      ...
    ], # Required. The input data of the node.
    "outputs": [
```

```
      {
        "data": <STRING>, # Required. The output data of the node.
      }
      , ...
    ], # Required. The output data of the node.
    "relatedFlowId": <LONG>, # Optional. The ID of the sub-flow to which the node belongs.
    "startEffectDate": <STRING>, # Optional. The start date of the time period during which scheduling is per
mitted.
    "endEffectDate": <STRING>, # Optional. The end date of the time period during which scheduling is perm
itted.
    "rerunAble": <BOOLEAN>, # Required. Specifies whether the node can be rerun.
    "userExtension": <STRING>, # Optional. The extended properties of the node.
    "childNodeId": <LONG>, # Optional. The ID of the child node. It is used when you add or delete a node de
pendency.
    "parentNodeId": <LONG>, # Optional. The ID of the parent node.
    "createInstanceUrl": <STRING>, # Optional. The URL used to create a node instance.
    "dqcType": <INTEGER>, # Optional. The type of the Data Quality check.
    "dqcDescription": <STRING>, # Optional. The description of the Data Quality check.
    "taskRerunTime": <INTEGER>, # Optional. The time when the node reruns.
    "taskRerunInterval": <LONG>, # Optional. The interval at which the node reruns.
    "manualTrigger": <INTEGER>, # Optional. Specifies whether the node needs to be manually triggered.
    "sycRefreshTask": <BOOLEAN>, # Optional. Specifies whether the node instances are updated when the
node is updated.
    "flowId": <LONG> # Required. The ID of the flow to which the node belongs.
  },
  ...
]
```

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "returnValue": <STRING> # requestId
}
```

2.5.4. Delete a node

URL: /v1.0/node/{projectEnv}/sync

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": DELETE_NODE_SYNC, # The operation type.
  "nodeId": <LONG>, # The node ID.
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": {
    "Node ID": "success",
    ...
  }
}
```

2.5.5. Query parent nodes

Description: You can call this operation to query the parent nodes of the specified node.

URL: /v1.0/node/{projectEnv}

Method: GET

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	(Required) The operation that you want to perform. Set the value to SEARCH_NODE_PARENTS_BY_DEPTH.
nodeId	Long	(Required) The ID of the node.
depth	Integer	(Required) The number of levels to be traversed.
nodeIds	String	(Optional) The node IDs, separated by commas (,).
cloudUUID	Long	(Optional) The node ID that is stored in the code library.

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": [<NodeEntity>, ...]
}
```

2.5.6. Query child nodes

Description: You can call this operation to retrieve child nodes of a specified node by level.

URL: /v1.0/node/{projectEnv}

Method: GET

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	(Required) The operation that you want to perform. Set the value to SEARCH_NODE_PARENTS_BY_DEPTH.
nodeId	Long	(Required) The node ID.

Name	Type	Description
depth	Integer	(Required) The number of levels to be traversed.
nodeIds	String	(Optional) The node IDs, separated by commas (,).
cloudUUID	Long	(Optional) The node ID that is stored in the code library.

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": [<NodeEntity>, ...]
}
```

2.5.7. Query the node code

URL: /v1.0/node/{projectEnv}

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_NODE_CODE.
cloudUUID	Long	The node ID that is stored in the code library.

Parameter	Type	Description
version	Long	The version of the node code.

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is successful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": <STRING> # The node code.
}
```

2.5.8. Query nodes

URL: /v1.0/node/{projectEnv}

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH.
nodeName	String	Optional. The name of the node.
cloudUUID	Long	Optional. The node ID that is stored in the code library.
nodeId	Long	Optional. The ID of the node to query.
version	Long	Optional. The version of the node code.

Parameter	Type	Description
inputs	String	Optional. The input data of the node. Separate multiple records with commas (,).
outputs	String	Optional. The output data of the node. Separate multiple records with commas (,).
fileId	Long	Optional. The ID of the code file.
nodeIds	String	Optional. The IDs of nodes to query. Separate multiple IDs with commas (,).
searchText	String	Optional. The characters to be entered on the front end used for searches, including the node ID and node name. The suffix fuzzy search is supported.
prgType	String	Optional. The type of the node code.
prgTypes	Integer	Optional. The types of the node code. Separate multiple types with commas (,).
appId	Long	Optional. The ID of the workspace to which the nodes belong. If you do not specify this parameter, all the applications under the current account are searched.
owner	String	Optional. The owner of the node.
createUser	String	Optional. The user who created the node.
modifyUser	String	Optional. The user who modified the node.
createTime	String	Optional. The time when the node was created, in the format of yyyy-MM-dd HH:mm:ss.
modifyTime	String	Optional. The time when the node was modified, in the format of yyyy-MM-dd HH:mm:ss.
baseLineId	Long	Optional. The ID of the baseline.
deployDate	String	Optional. The time when the node was deployed.
orderBy	String	Optional. Specifies the way in which the nodes are ordered.
isDetail	Boolean	Optional. Specifies whether to query the node details.
depth	Integer	Optional. The number of levels to be traversed.
flowId	Long	The ID of the flow to which the node belongs.

Parameter	Type	Description
nodeType	Integer	The type of the node. Valid values: 0, 1, 2, and 3. 0: the nodes that run as scheduled. 1: a node that must be manually triggered and cannot be scheduled. 2: a paused node. Paused nodes are initiated as scheduled but the system immediately returns a failure indication without executing the nodes. 3: a dry-run node. Dry-run nodes are initiated as scheduled but the system immediately returns a success indication without executing the nodes. This parameter is optional.
tenantId	Long	Optional. The ID of the tenant.
bizDate	String	Optional. The date-based timestamp of the node, in the format of yyyy-MM-dd.
prgName	String	Optional. The name of the node code file.
isOnline	Integer	Optional. The online status of the node. Valid values: 0 and 1. 0: online. 1: offline.
modifyStartTime	String	Optional. The start of the time period during which the node was modified.
modifyEndTime	String	Optional. The end of the time period during which the node was modified.
deployStartingDate	String	Optional. The start of the time period during which the node was deployed.
deployDeadlineDate	String	Optional. The end of the time period during which the node was deployed.
nodeForm	String	The source of the node.
rerunAble	Boolean	Optional. Specifies whether to query nodes that can be or cannot be rerun.
bizId	Long	Optional. The workflow ID.
solId	Long	Optional. The solution ID.
filePaths	String	Optional. The paths to the node code files.
isSmoke	Boolean	Optional. Specifies whether to query nodes for which the smoke test is or is not run.
resGroupId	Long	Optional. The ID of the resource group.

Parameter	Type	Description
isDeploy	Boolean	Optional. Specifies whether to query nodes that are deployed or are not deployed.
deployType	Integer	Optional. The type of the node after deployment. Valid values: 0, 1, 2, and 3. The value 0 specifies no type limit. The value 1 specifies a new node. The value 2 specifies a modified node. The value 3 specifies a deleted node.
pageStart	Integer	None.
pageSize	Integer	None.

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": [
    <NodeEntity>,
    ...
  ]
}
```

2.5.9. Query a node by ID

URL: /v1.0/node/{projectEnv}

Method: GET

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	SEARCH_NODE_BY_ID
nodeId	Long	None.

Name	Type	Description
isDetail	Boolean	Indicates whether to query the details about the node.

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", #  0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": <NodeEntity>
}
```

2.5.10. Obtain the total number of nodes

URL: /v1.0/node/{projectEnv}

Method: GET

Headers: application/json; charset=UTF-8

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_NODE_COUNT.

Response

```
{
  "returnCode": "0",
  "returnErrorSolution": "",
  "returnMessage": "",
  "requestId": null,
  "returnValue": 283232, # The total number of nodes.
  "success": true
}
```

2.5.11. Query node instances

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH.
appId	Long	The ID of the workspace to which the node instance belongs.
appIds	String	The IDs of workspaces to which the node instances belong. Separate multiple application IDs with commas (,).
taskId	Long	The ID of the node instance to query.
nodeId	Long	The ID of the node to which the node instance belongs.
inGroupId	Long	The ID of the cycle to which the node instance belongs.
nodeIds	String	The IDs of nodes to which the node instances belong. Separate multiple node IDs with commas (,).
dagId	Long	The ID of the workflow instance to which the node instance belongs.

Parameter	Type	Description
taskType	Integer	The type of the node instance. Valid values: 0, 1, 2, and 3. • 0: routine. • 1: manually triggered. • 2: paused. • 3: dry-run.
taskTypes	String	The types of the node instances. Separate multiple types with commas (,).
taskStatuses	String	The statuses of the node instances. Separate multiple statuses with commas (,).
dagType	Integer	The type of the workflow instance to which the node instance belongs. Valid values: 1, 2, 3, and 4. • 1: the routine workflow instance that is run as scheduled. • 2: the workflow instance for smoke test. • 3: the workflow instance for generating retroactive data. • 4: the manually triggered workflow instance.
prgType	Integer	The type of the node code.
status	Integer	The status of the node.
owner	String	The owner of the node or the node instance.
bizdate	String	The date-based timestamp.
odpsInstanceId	String	The ID of the ODPS node instance.
bizBeginHour	String	The start of the time period when the workflow runs.
bizEndHour	String	The end of the time period when the workflow runs.
beginBizDate	String	The start date of the period when the workflow runs.
endBizDate	String	The end date of the period when the workflow runs.
searchText	String	The characters entered on the front end used for searches, including the node ID and node name.
baseLineId	Long	The ID of the baseline to which the node instance belongs.
createTime	String	The time when the node instance was created.
isDetail	Boolean	Optional. Specifies whether to query the details of the node instance.
opSeq	Long	The sequence number of the operation.
depth	Integer	Optional. The number of levels to be traversed.

Parameter	Type	Description
historyId	Long	The ID of the node instance log.
costTime	Integer	The time elapsed for running the node instance. Unit: seconds.
startRunTimeFrom	String	The start of a time period during which the node instance starts running.
startRunTimeTo	String	The end of a time period during which the node instance starts running.
finishRunTimeFrom	String	The start of a time period during which the node instance stops running.
finishRunTimeTo	String	The end of a time period during which node instance stops running.
createTimeFrom	String	The start of a time period during which the node instance was created.
createTimeTo	String	The end of a time period during which the node instance was created.
prgTypes	String	The types of the node code. Separate multiple types with commas (,).
taskIds	String	The IDs of the node instances to query.
rerunAble	Boolean	Optional. Specifies whether to query node instances that can be or cannot be rerun.
dagName	String	The name of the workflow instance.
orderBy	String	Optional. Specifies the way in which the node instances are ordered.
bizId	Long	The ID of the workflow.
solId	Long	The ID of the solution.
gmtdate	String	The date when the node instance was generated.
createUser	String	The user who created the node instance.
modifyUser	String	The user who modified the node instance last time.
modifyTime	String	The time when the node instance was last modified.
deployDate	String	The date when the node instance was deployed.
pageStart	Integer	The start number of the returned page.
pageSize	Integer	The number of entries returned per page.

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": [
    <TaskEntity>,
    ...
  ]
}
```

2.5.12. Query the operational logs of a node instance

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_TASK_RUNLOG.
taskId	Long	The ID of the node instance. Specify either taskId or historyId.
historyId	Long	The ID of the node instance log. Specify either taskId or historyId.

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "returnValue": <STRING>
}
```

2.5.13. Query a node instance by ID

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_TASK_BY_ID.
taskId	Long	The ID of the node instance to query.
isDetail	Boolean	Specifies whether to obtain the application details.

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "returnValue": <TaskEntity>
}
```

2.5.14. Query the parent node instance by level

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_TASK_PARENTS_BY_DEPTH.
taskId	Long	The ID of the business flow instance that contains the task instance.
depth	Integer	The level of the node.

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": [
    <TaskEntity>,
    ...
  ]
}
```

2.5.15. Query the child node instances at a specified level

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_TASK_CHILDREN_BY_DEPTH.
taskId	Long	The ID of the node instance.
depth	Integer	The level where the node instance resides.

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": [
    <TaskEntity>,
    ...
  ]
}
```

2.5.16. Query the number of node instances in each status of an application

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_TASK_STATISTICS.
appId	Long	-
bizdate	String	-

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": {
    "dagType": <INTEGER>,
    "appId": <LONG>,
    "bizdate": <STRING>, # The date-based timestamp.
    "noRun": <INTEGER>,
    "running": <INTEGER>,
    "waittingResource": <INTEGER>,
    "failure": <INTEGER>,
    "success": <INTEGER>,
    "waittingTime": <INTEGER>
  }
}
```

2.5.17. Query the number of task instances in each status on a specified business date

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_TASK_STATISTICS_BY_OWNER.
bizdate	String	None.

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
   "returnCode": "0", # 0 indicates that the operation is called.
   "returnMessage": <STRING>, # The operation details.
   "successCode": "0",
   "count": <INTEGER>,
   "returnValue": {
      "dagType": <INTEGER>,
      "appId": <LONG>,
      "bizdate": <STRING>, # The business date.
      "noRun": <INTEGER>,
      "running": <INTEGER>,
      "waittingResource": <INTEGER>,
      "failure": <INTEGER>,
      "success": <INTEGER>,
     "waittingTime": <INTEGER>
   }
}
```

2.5.18. Query the number of node instances in each status for applications

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Query parameters in the request URL

Parameter	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_TASK_SUCCESSINFO.
bizdate	String	The date-based timestamp.
costTime	Integer	The time elapsed for running the node instance. Unit: seconds.
startRunTimeFrom	String	The start of a time period during which the node instance starts running.
startRunTimeTo	String	The end of a time period during which the node instance starts running.
finishRunTimeFrom	String	The start of a time period during which the node instance stops running.
finishRunTimeTo	String	The end of a time period during which the node instance stops running.
createTimeFrom	String	The start of a time period during which the node instance was created.
createTimeTo	String	The end of a time period during which the node instance was created.
appId	Long	The ID of workspace to which the node instance belongs.
searchText	String	The characters entered on the front end used for searches, including the node ID and node name. The suffix fuzzy search is supported.
nodeId	Long	The ID of the node to which the node instance belongs.
nodeIds	String	The IDs of nodes to which the node instances belong.
taskIds	String	The IDs of the node instances to query.
taskStatuses	String	The statuses of the node instances to query.

Parameter	Type	Description
taskTypes	String	The types of the node instances to query.
prgTypes	String	The time when the node instance was created.
createTime	String	
modifyTime	String	The time when the node instance was last modified.
deployDate	String	The date when the node instance was deployed.
gmtdate	String	The date when the node instance was generated.
bizBeginHour	String	The start of the time period when the workflow runs.
bizEndHour	String	The end of the time period when the workflow runs.
bizBeginDate	String	The start date of the period when the workflow runs.
bizEndDate	String	The end date of the period when the workflow runs.

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "returnValue": {
    "totalCount": <INTEGER>,
    "notRunCount": <INTEGER>,
    "waitTimeCount": <INTEGER>,
    "waitResCount": <INTEGER>,
    "runningCount": <INTEGER>,
    "failureCount": <INTEGER>,
    "successCount": <INTEGER>
  }
}
```

2.5.19. Rerun an instance

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "RERUN_TASK",
  "taskId": <LONG>
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.20. Restore an instance

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "RESUM_TASK",
  "taskId": <LONG>
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # The value 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.21. Set the instance status to Success and trigger descendant instances

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "SETSUCCESS_TASK",
  "taskId": <LONG>
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.22. Terminate a task

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "KILL_TASK",
  "taskId": <LONG>
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.23. Rerun a descendant instance

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "FIXDATA",
  "taskId": <LONG>,
  "includeTaskIds": [<LONG>, ...]
}
```

Response

```
{
   "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.24. Update the scheduling configurations of an instance

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "Executemethod": "maid ",
  "taskId": <LONG>
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.25. Pause an instance

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "STOP_TASK",
  "taskId": <LONG>
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.26. Terminate multiple instances

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "BATCH_KILL_TASK",
  "includeTaskIds": [<LONG>, ...]
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.27. Rerun multiple instances

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
   "executeMethod": "BATCH_RESUM_TASK",
   "includeTaskIds": [<LONG>, ...]
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.28. Resume multiple tasks

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
    "executeMethod": "BATCH_RESUM_TASK",
    "includeTaskIds": [<LONG>, ...]
}
```

Response

```
{
   "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
   "returnCode": "0", # 0 indicates that the operation is called.
   "returnMessage": <STRING>, # The operation details.
   "successCode": "0",
   "returnValue": true
}
```

2.5.29. Pause multiple instances

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
    "executeMethod": "BATCH_STOP_TASK",
    "includeTaskIds": [<LONG>, ...]
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.30. Set the status of multiple instances to Success

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
  "executeMethod": "BATCH_SETSUCCESS_TASK",
  "includeTaskIds": [<LONG>, ...]
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.31. Retrieve the total number of instances on the day

URL: /v1.0/task/{projectEnv}

Headers: application/json; charset=UTF-8

Method: GET

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	SEARCH_TASK_STATISTICS_BY_DATE
dagType	Integer	(Optional) Default value: 1.
bizdate	String	(Optional) The day before the current day, in the format of yyyy-MM-dd.

Response

```
{
  "returnCode": "0",
  "returnErrorSolution": "",
  "returnMessage": "",
  "requestId": null,
  "Redding value": 283232, # The number of instances.
  "success": true
}
```

2.5.32. Query business flow instances

URL: /v1.0/dag/{projectEnv}

Method: GET

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

URL query string parameters

Name	Type	Description
executeMethod	String	The operation that you want to perform. Set the value to SEARCH_DAG_BY_ID.
dagId	Long	The ID of the business flow instance that contains the task instance.

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": [<DagEntity>, ...]
}
```

2.5.33. Retroactive execution

URL: /v1.0/dag/{projectEnv}

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
    "executeMethod": "CREATE_DAG",
    "name": <STRING>, # The name of the retroactive workflow.
 "type": 3, # The business flow type. 3: retroactive execution.
 "rootNodeId": <LONG>, # The root node ID from which the retroactive execution starts. If data of only one n
ode is retroacted, set the value to the ID of the node.
 "includeNodeIds": [<LONG>, ...], # The IDs of the nodes to be retroacted. If data of only one node is retroacte
d, set the value to the ID of the node.
 "excludeNodeIds": [<LONG>, ...], # The IDs of nodes that are excluded from the retroactive execution.
 "startBizDate": <STRING>, # The start date-based timestamp of data involved in the retroactive task, in the f
ormat of yyyy-MM-dd hh:mm:ss.
 "endBizDate": <STRING>, # The end date-based timestamp of data involved in the retroactive task, in the fo
rmat of yyyy-MM-dd hh:mm:ss.
 "bizBeginTime": <STRING>, # The start time of the hourly task. For example, 13:04:04.
 "bizEndTime": <STRING>, # The end time of the hourly task. For example, 14:04:04.
 "isParallel": <BOOLEAN>, # Indicates whether the DAGs of retroactive execution can run concurrently.
 "nodeParas": {<LONG>: <STRING>, ...}, # (Optional) The user-defined parameter for updating a node. The ke
y is node ID. The value is the ID of the node.
    "dagPara": <STRING>, # (Optional) The parameter for configuring a DAG, which applies to all instances in
the DAG.
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": <LONG>, # The DAG ID.
}
```

2.5.34. Run a smoke test

URL: /v1.0/dag/{projectEnv}

Method: POST

Path parameters

Parameter	Type	Description
projectEnv	String	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.

Body

```
{
    "executeMethod": "CREATE_DAG",
    "name": <STRING>, # The custom name of the workflow.
 "type": 2, # The type of the workflow instance. The value 2 specifies a smoke test.
 "bizdate": <STRING>, # The date-based timestamp, in the format of yyyy-MM-dd hh:mm:ss.
 "rootNodeId": <LONG>, # The ID of the root node.
 "rootNodeAppId": <LONG>, # The ID of the workspace to which the root node belongs.
 "includeNodeIds": [], # The IDs of the whitelisted nodes. Leave this parameter blank.
 "excludeNodeIds": [], # The IDs of the blacklisted nodes. Leave this parameter blank.
 "bizBeginTime": <STRING>, # The start time of the hourly node, for example, 13:04:04.
 "bizEndTime": <STRING>, # The end time of the hourly node, for example, 14:04:04.
 "isParallel": false, # Specifies whether the node instance for running the smoke test can be run concurrently
.
 "nodeParas": {<LONG>: <STRING>, ...}, # Optional. The custom key-value pairs for updating a node. Set the k
eys to node ID and the values to the scheduling parameters of the node.
  "dagPara": <STRING>, # Optional. The key-value pair used to generate retroactive data for a workflow inst
ance. It applies to all node instances that belong to the workflow instance.
}
```

Response

```
{
  "requestId": <STRING>, # The request ID. It is used to find the log entries for troubleshooting.
  "returnCode": "0", # The execution status of the operation. The value 0 indicates that the operation is succ
essful.
  "returnMessage": <STRING>, # The returned execution details.
  "successCode": "0",
  "returnValue": <LONG>, # dag id
}
```

2.5.35. Execute an ad-hoc business flow

URL: /v1.0/dag/{projectEnv}

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
    "executeMethod": "CREATE_BUS_PROC_DAG",
    "name": <STRING>, # The name of the ad-hoc business flow instance.
 "type": 5, # Set the type of the ad-hoc business flow instance to 5.
    "flowId": <LONG>, # The ID of the ad-hoc business flow.
 bizdate": <STRING>,  # The date-based timestamp of data, in the format of yyyy-MM-dd hh:mm:ss.
 "rootNodeAppId": <LONG>,  # The ID of the application.
 "includeNodeIds": [],
 "excludeNodeIds": [],
 "isParallel": false,
 "nodeParas": {<LONG>: <STRING>, ...}, # (Optional) The user-defined parameter for updating a node. The ke
y is node ID. The value is the ID of the node.
    "dagPara": <STRING>, # (Optional) The parameter for configuring a DAG, which applies to all instances in
the DAG.
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "count": <INTEGER>,
  "returnValue": <LONG>, # The DAG ID.
}
```

2.5.36. Terminate a business flow

URL: /v1.0/dag/{projectEnv}

Method: POST

Path parameters

Name	Type	Description
projectEnv	String	Valid values: DEV and PROD.

Request body

```
{
    "executeMethod": "CREATE_DAG",
   "Dagids": [<long>,...], # The ID of the DAGs to be terminated.
}
```

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "Charlevalue": <Boolean>
}
```

2.5.37. Submit a resource package

Description: You can call this operation to submit a resource package that contains flow and node definitions and resources. After the operation is called, RequestID is returned.

URL: /submission/project/{projectIdentifier}

Method: POST

Headers: Content-type: multipart/form-data

Data type: zipped files

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": true
}
```

2.5.38. Query the submission status of a resource package

URL: /submission/project/{projectIdentifier}/{requestId}

Path parameters

Name	Type	Description
requestId	String	None.

Response

```
{
  "requestId": <STRING>, # The request ID, used to search for logs to diagnose the issues.
  "returnCode": "0", # 0 indicates that the operation is called.
  "returnMessage": <STRING>, # The operation details.
  "successCode": "0",
  "returnValue": {
    "createTime": <STRING>, # The time at which the submitted request is accepted by the system.
    "name": <STRING>, # The generated name in the [the caller system name-backToCheckId-createTime] format.
    "status": <STRING>, # The status. Values: success, fail, and running.
    "message": <STRING>
  }
}
```

2.5.39. Data structure

2.5.39.1. NodeModifyDto

```
{
    "nodeId": <LONG>, # The ID of the node in the scheduling system. This parameter must be specified when you call this operation to update a node.
    "uniqueId": <LONG>, # Required. The unique ID of the request. It is transferred from the client and used only by the client to compare execution results.
    "nodeName": <STRING>, # Required. The name of the node.
    "nodeType": <INTEGER>, # Required. The node type. Valid values: 0, 1, 2, and 3. The value 0 specifies a node that runs as scheduled. The value 1 specifies a one-off node. The value 2 specifies a paused node. The value 3 specifies a dry-run node.
    "prgType": <INTEGER>, # Required. The type of the node code, for example, odps_shell and odps_sql.
    "description": <STRING>, # Optional. The description of the node.
```

```
    "paraValue": <STRING>, # Optional. The key-value pairs of a node.
    "priority": <INTEGER>, # Optional. The priority of the node. Default value: 0.
    "cronExpress": <STRING>, # Required. The CRON expression.
    "cloudUUID": <LONG>, # The node ID that is stored in the code library. This parameter is required when y
ou call this operation to update a node and is optional when you call this operation to create a node.
    "fileId": <LONG>, # Optional. The ID of the code file.
    "fileVersion": <LONG>, # Optional. The version number of the code file.
    "owner": <STRING>, # Required. The owner of the node.
    "resGroupId": <LONG>, # Required. The ID of the resource group to which the node belongs.
    "appId": <LONG>, # Required. The ID of the workspace to which the node belongs.
    "baseLineId": <LONG>, # The ID of the baseline to which the node belongs.
    "createUser": <STRING>, # Required. The user who created the node.
    "modifyUser": <STRING>, # Required. The user who modified the node last time.
    "cycType": <INTEGER>, # Required. The cycle type. Valid values: 0, 1, and 2. The value 0 specifies that the
cycle is equal to or longer than one day. The value 1 specifies that the cycle is shorter than one day. The valu
e 2 specifies that the cycle is shorter than one day and node instances are executed in order of the node inst
ance IDs, such as 1, 2, 3...
    "dependentType": <INTEGER>, # Required. The dependency type of the node. Valid values: 0, 1, 2, and 3.
The value 0 specifies that the node can be run without waiting for the previous cycle of a specified node is co
mpleted. The value 1 specifies that the node is run only after the previous cycle of a custom node is complet
ed. The value 2 specifies that the node is run only after the previous cycle of its child nodes is completed. Th
e value 3 specifies that the node is run only after its previous cycle is completed.
    "dependentNodeIds": [<LONG>, ...], # The IDs of the custom cross-version dependent nodes. This param
eter must be specified when the value of dependentType is set to 3. Separate multiple IDs with commas (,).
    "multiInstKillType": <INTEGER>, # Optional. The type of instances to be terminated. Valid values: 0, 1, an
d 2. The value 0 specifies to terminate no node instances. The value 1 specifies to terminate non-routine nod
e instances. The value 2 specifies to terminate routine node instances.
    "refreshToTask": <BOOLEAN>, # Optional. Specifies whether the configuration takes effect on the curren
t node instance.
    "updateNodeOnly": <BOOLEAN>, # Specifies whether to refresh the metadata of ancestor and descenda
nt nodes. Default value: false.
    "codeSrc": <STRING>, # The node code. This parameter must be specified when the value of updateNode
Only is set to false.
    "envType": <INTEGER> , # Optional. The type of the runtime environment.
    "nodeFrom": <STRING>, # Optional. The source of the node, that is, the system from which the node is d
eployed.
    "inputs": [
      {
        "data": <STRING>, # Required. The input data of the node.
      },
      ...
```

```
    ], # Required. The input data of the node.
    "outputs": [
      {
        "data": <STRING>, # Required. The output data of the node.
      }
      , ...
    ], # Required. The output data of the node.
    "relatedFlowId": <LONG>, # Optional. The ID of the sub-flow to which the node belongs.
    "startEffectDate": <STRING>, # Optional. The start date of the time period during which scheduling is per
mitted.
    "endEffectDate": <STRING>, # Optional. The end date of the time period during which scheduling is perm
itted.
    "rerunAble": <BOOLEAN>, # Required. Specifies whether the node can be rerun.
    "userExtension": <STRING>, # Optional. The extended properties of the node.
    "childNodeId": <LONG>, # Optional. The ID of the child node. It is used when you add or delete a node de
pendency.
    "parentNodeId": <LONG>, # Optional. The ID of the parent node.
    "createInstanceUrl": <STRING>, # Optional. The URL used to create a node instance.
    "dqcType": <INTEGER>, # Optional. The type of the Data Quality check.
    "dqcDescription": <STRING>, # Optional. The description of the Data Quality check.
    "taskRerunTime": <INTEGER>, # Optional. The time when the node reruns.
    "taskRerunInterval": <LONG>, # Optional. The interval at which the node reruns.
    "manualTrigger": <INTEGER>, # Optional. Specifies whether the node needs to be manually triggered.
    "sycRefreshTask": <BOOLEAN>, # Optional. Specifies whether the node is a synchronous reloading node.

    "flowId": <LONG> # Required. The ID of the flow to which the node belongs.
  }
```

2.5.39.2. TaskEntity

```
private Long id; // The node ID recorded in the database. It increments automatically.
private Long taskId; // The ID of the node instance.
private Long nodeId; // The ID of the node to which the node instance belongs.
private Long dagId; // The ID of the workflow instance to which the node instance belongs.
private Integer inGroupId; // The sequence number of the cycle on the current day of the hourly node.
private Integer taskType; // The type of the node. Valid values: 0, 1, 2, and 3. The value 0 specifies a node that
runs as scheduled. The value 1 specifies a one-off node. The value 2 specifies a paused node. The value 3 spe
cifies a dry-run node.
private Integer dagType; // The type of the workflow instance to which the node instances belong. Valid valu
es: 1, 2, 3, and 4. The value 1 specifies the routine workflow instance that is run as scheduled. The value 2 sp
```

```
ecifies the workflow instance for smoke test. The value 3 specifies the workflow instance for generating retr
oactive data. The value 4 specifies the manually triggered workflow instance.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date dueTime; // The time when the node is scheduled to run.
private Integer status; // The status of the node.
private Long opSeq; // The sequence number of the operation.
private Integer opCode; // The operation command.
private String owner; // The owner of the node instance.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date bizdate; // The date-based timestamp.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date gmtdate; // The date when the node instance was generated.
private String gateway; // The gateway server on which the node runs.
private String gwProcessId; // The ID of the gateway process in which the node runs.
private String gwLogFile; // The log file of the gateway on which the node runs.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date alisaReturnTime; // The date when the alisa service is rerun.
private String prgName; // The name of the node code file.
private Integer prgType; // The type of the node code, for example, odps_sql.
private Integer priority; // The priority of the node.
private Integer weight; // The weight of the node.
private String execName; // The name of the node code file.
private Long fileId; // The ID of the node code file.
private Integer fileVersion; // The version number of the node code file.
private String odpsProjectName; // The name of the MaxCompute project.
private String paraValue; // The key-value pairs of a node.
private String gwLogLocalFile; // The local log file of the gateway on which the node runs.
private String resGroupIdentifier; // The identifier of the resource group.
private Long resGroupId; // The ID of the resource group to which the node instance belongs.
private Long baseLineId; // The ID of the baseline to which the node instance belongs.
private Long appId; // The ID of the workspace to which the node instance belongs.
private String appName; // The name of the application to which the node instance belongs.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date beginWaitTimeTime; // The time elapsed until the node status is changed to Pending (Schedule)
.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
```

```
@JsonDeserialize(using = DateDeserializer.class)
private Date beginWaitResTime; // The time elapsed until the node status is changed to Pending (Resources)
.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date beginRunningTime; // The time elapsed until the node status is changed to Running.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date finishTime; // The end of the time period when the node instance runs.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date createTime; // The time when the node instance was created.
private String createUser; // The user who created the node instance.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date modifyTime; // The time when the node instance was modified last time.
private String modifyUser; // The user who modified the node instance last time.
private Integer multiInstCheckType; // The identifier of the operation that checks whether multiple instance
s exist.
private Integer multiKillType; // The identifier of the operation that terminates multiple instances.
private Integer cycType; // The cycle type.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date cycTime; // The period that the cycle lasts.
private Integer roleType; // The role of the node, for example, a leaf node or a non-leaf node.
private Integer dependentType; // The dependency type of the node.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date deployDate; // The date when the node instance was deployed.
private String nodeName; // The name of the node.
private Integer rerunTimes; // The number of times the node is rerun.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date nodeModifyTime; // The time when the node was modified last time.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date refreshTime;
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
@JsonDeserialize(using = DateDeserializer.class)
private Date delayExecTime;
private Boolean rerunAble;
```

```
private Boolean rerunAble;
private List<TaskRelationDto> parentTaskRelations; // The parent relations of the node instance.
private List<TaskRelationDto> childTaskRelations; // The child relations of the node instance.
private Integer isRunOver;// Indicates whether the node instance has been run.
private Long bizId;
private List<MetaTable> parentOutputTabMetas; // The metadata of the table on which the node instance d
epends.
private List<MetaTable> outputTabMetas; // The metadata of the output table of the node instance.
```

2.5.39.3. DagEntity

```
private String name; // The name of the workflow instance.
 private Integer type; // The type of the workflow instance.
 @JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
     @JsonDeserialize(using = DateDeserializer.class)
 private Date bizdate; // The date-based timestamp. This parameter must be specified for a one-off node.
 private Long rootNodeId; // The ID of the root node.
 private List<Long> includeNodeIds = new ArrayList<Long>(); // The IDs of the whitelisted nodes.
 private List<Long> excludeNodeIds = new ArrayList<Long>(); // The IDs of the blacklisted nodes.
 @JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
     @JsonDeserialize(using = DateDeserializer.class)
 private Date startBizDate; // The start of the time period when the retroactive data generation task runs.
 @JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
     @JsonDeserialize(using = DateDeserializer.class)
 private Date endBizDate; // The end of the time period when the retroactive data generation task runs.
 private String bizBeginTime; // The start time of the hourly node, for example, 13:04:04.
 private String bizEndTime; // The end time of the hourly node, for example, 14:04:04.
 private Boolean isParallel; // Specifies whether the workflow instance to which the retroactive data generat
ion task belongs can be run concurrently.
 // ////////////////////////////////The preceding parameters are required for creating an ad-hoc workflow.//////////////
////////////////////////////
 private Long dagId; // The ID of the workflow instance.
 private Integer status; // The status of the workflow instance.
 @JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
     @JsonDeserialize(using = DateDeserializer.class)
 private Date gmtdate; // The date when the workflow instance was generated.
 @JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
     @JsonDeserialize(using = DateDeserializer.class)
 private Date startDate; // The date when the workflow instance starts.
 @JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
     @JsonDeserialize(using = DateDeserializer.class)
```

```
private Date startTime; // The start of the time period when the workflow instance runs.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
    @JsonDeserialize(using = DateDeserializer.class)
private Date finishTime; // The end of the time period when the workflow instance runs.
private String rootTaskIds; // The IDs of instances that belong to the root node.
private Long appId; // The ID of the workspace to which the workflow instance belongs.
private Long parentDagId; // The ID of the parent workflow instance.
private Long childDagId; // The ID of the child workflow instance.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
    @JsonDeserialize(using = DateDeserializer.class)
private Date createTime; // The time when the workflow instance was created.
private String createUser; // The user who created the workflow instance.
@JsonSerialize(using = DateSerializer.class, include = Inclusion.NON_NULL)
    @JsonDeserialize(using = DateDeserializer.class)
private Date modifyTime; // The time when the workflow instance was modified last time.
private String modifyUser; // The user who modified the workflow instance last time.
private Integer createStatus; // The status of the creation. Valid values: 1, 2, 3, and 4. 1: The instance is being
initiated. 2: The instance is being created. 3: The instance is created. 4: The instance failed to be created.
private Long opSeq;
private Integer dagNum;
private Boolean isDependDaily = false; // Specifies whether the workflow instance depends on an instance i
n the development environment.
```

2.5.39.4. TaskStatus

- Not_run (1, "The task is not running "),
- Wait_time (2, "The task is pending execution."),
- Wait_resource (3, "The task is waiting for resources."),
- RUNNING(4, "The task is running."),
- FAILURE(5, "An error occurred while the task is executed."),
- SUCCESS(6, "The task execution is completed.");

2.5.39.5. Dagstatus

- Created (1, "The DAG has been created."),
- RUNNING(4, "The DAG is running."),
- FAILURE(5, "An error occurred while the DAG is executed."),
- SUCCESS(6, "The DAG is executed.");

2.5.39.6. ProgramType

- IDE_SHELL(6),

- ODPS_SQL(10),
- ODPS_MR(11),
- POL_SYNC(23) Synchronization task

2.5.39.7. CycleType

- DAY(0), // Indicates cycles that are no shorter than one day.
- NOT_DAY(1), // Indicates cycles that are shorter than one day.
- NOT_DAY_SEQ(2); // The sequence number of each cycle that is shorter than one day. The cycles are arranged in an ordered array.

2.5.39.8. DependencyType

- NONE(0), // The node can be run without waiting for the previous cycle of a specified node is completed.
- USER_DEFINE(1), // The node is run only after the previous cycle of a custom node is completed.
- CHILD(2), // The node is run only after the previous cycle of its child nodes is completed.
- SELF(3); // The node is run only after its previous cycle is completed.

2.6. Java SDK

2.6.1. GetInstanceSummary

You can call this operation to query the information about instances of readers or writers based on the specified Apsara Stack tenant account ID.

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to GetInstanceSummary.
Aliuid	String	Yes	The ID of the Apsara Stack tenant account ID.
EndTime	String	Yes	The end time of time range where this operation is performed. Specify the time in the yyyy-mm-dd format.
StartTime	String	Yes	The start time of time range where this operation is performed. Specify the time in the yyyy-mm-dd format.

Parameter	Type	Required	Description
Type	String	Yes	Specifies whether to return the instance information about readers or writers. A value of 1 indicates that the result is the information about instances of readers. A value of 2 indicates that the result is the information about instances of writers.

Response parameters

Parameter	Type	Description
Data	Long	The information about instances.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample request

```
http(s)://[Endpoint]/? Action=GetInstanceSummary
&Aliuid=178*****2537
&EndTime=2019-02-21
&StartTime=2019-02-21
&Type=1
&<Common request parameters>
```

Sample response

```
{
"Data": [
{
"InstanceCnt": 17,
"AliyunKp": "1782****62537",
"PluginName": "mysql", "TotalRecords": 4255624,
"TotalBytes": 266409932
}
],
"ErrMsg": "success",
"RequestId": "e19f8502-76d2-4cb1-ab54-b93013cfad17", "ErrCode": 0
}
```


2.6.2. AddResGroupGateWay

You can call this operation to add a gateway to a resource group.

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to AddResGroupGateWay.
BaseId	String	Yes	The ID of the Apsara Stack account.
Identifier	String	Yes	The identifier of the resource group.
Memory	Integer	Yes	The memory size of the resource group.
NodeAddress	String	Yes	The IP address of the node that runs the gateway.
NodeName	String	Yes	The name of the node that runs the gateway.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
TenantId	Long	Yes	The ID of the tenant.

Parameter	Type	Required	Description
VCpu	Integer	Yes	The number of cores in the vCPU of the gateway.

Response parameters

Parameter	Type	Description
Data	Boolean	The returned data. Indicates whether the operation is successful.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=AddResGroupGateWay
&BaseId=1486***73474
&Identifier=3907f9a58******6b2506
&Memory=3
&NodeAddress=12.12.12.12
&NodeName=hostname
&ProjectId=75059
&TenantId=2678***8690
&VCpu=8
&<Common request parameters>
```

Sample success responses

```
{
  "requestId": "0bc1748b*****493e5f15",
  "data": true,
  "errMsg": "success",
  "errCode": 0
}
```

2.6.3. CreateConnection

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to CreateConnection.
Connection	String	Yes	The connection string of the data source.
Name	String	Yes	The name of the data source.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
Type	String	Yes	The type of the data source.
EnvType	Integer	No	The type of the environment. Valid values: 0 and 1. The value 0 specifies the development environment. The value 1 specifies the production environment.
TenantId	Long	No	The ID of the tenant.
Shared	Boolean	No	Specifies whether the data source is shared or not.
Description	String	No	The description of the data source.
SubType	String	No	The subtype of the data source. Currently, only MySQL is supported.

Response parameters

Parameter	Type	Description
Data	Long	The returned data. If the operation is successful, it returns the ID of the connected data source. If the operation fails, it returns null.

Parameter	Type	Description
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=CreateConnection
&<Common request parameters>
```

2.6.4. CreateDag

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to CreateDag.
AccountId	String	Yes	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	Yes	The password of the account that is assigned by the scheduling system to call this operation.
ClientName	String	Yes	The name of the client that is assigned by the scheduling system to call this operation.
BaseId	String	Yes	The ID of the Apsara Stack account that you use to call this operation.

Parameter	Type	Required	Description

DagModifyDto	String	Yes	{
			"executeMethod": "CREATE_DAG",
			"name": "test_create_dag",
			"type": 3,
			"includeNodeIds": [3231],
			"excludeNodeIds": [],
			"rootNodeId": 3231,
			"startBizDate": "2018-07-08 00:00:00",
			"endBizDate": "2018-07-09 00:00:00",
			"isParallel": false
			}

Parameter	Type	Required	Description
ProjectEnv	String	Yes	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	Long	The value of the DagID parameter.

Sample success responses

```
{
  "returnCode":"0",
  "returnErrorSolution":"",
  "returnMessage":"",
  "requestId":"740ded9e-f6f5-4fde-a419-9b54***e8",
  "returnValue":300000119,
  "success":true
}
```

2.6.5. CreateDQCEntity

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to CreateDQCEntity.
EnvType	String	Yes	The type of the computing engine.
MatchExpression	String	Yes	The name of the partition filter expression.
ProjectName	String	Yes	The name of the MaxCompute project.
TableName	String	Yes	The name of the table.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code. The value 0 indicates that the operation is successful. Other values indicate that the operation failed.
ReturnValue	Integer	The ID of the partition filter expression that has been created.

Sample requests

```
/? Action=CreateDQCEntity
&EntityLevel=0
&EnvType=odps
&MatchExpression=ds=${yyyymmdd-1}
&ProjectName=autotest
&TableName=test_dqc_decimal_1119_2
&<Common request parameters>
```

Sample success responses

```
{
"ReturnCode": "0",
"ReturnValue": 4003923
}
```

2.6.6. CreateDQCFollower

You can call this operation to subscribe to messages from Data Quality. The messages are generated for the partitions that are specified by the partition filter expression. The methods of receiving messages include the emails, Short Message Service (SMS), and DingTalk.

> **Note** To receive messages by using DingTalk, specify a DingTalk group that receives the messages on the Robot Management page and add a chatbot. After you add the chatbot, a webhook URL for the chatbot is automatically generated. You can then enable Data Quality to send HTTP POST requests to this URL to send messages to the specified DingTalk group.

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to CreateDQCFollower.
ProjectName	String	No	The name of the MaxCompute project.
Follower	String	No	The object of receiving the subscribed messages. If you want to receive the messages by using emails or the SMS, specify your employee ID. If you want to receive the messages by using DingTalk, specify the webhook URL.

Parameter	Type	Required	Description
AlarmMode	Integer	No	The method of receiving alerts and notifications. The value 1 specifies that you receive alerts and notifications by using emails. The value 2 specifies that you receive alerts and notifications by using emails and the SMS. The value 3 specifies that you receive alerts and notifications from the specified DingTalk group by using the webhook URL. The value 4 specifies that you receive alerts and notifications from the specified DingTalk group and mention all members in the group.
EntityId	Long	No	The ID of the partition filter expression.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	Integer	The ID of the subscription.

Sample requests

```
/? Action=CreateDQCFollower
&AlarmMode=2
&EntityId=4003922
&Follower=50624
&ProjectName=autotest
&<Common request parameters>
```

Sample success responses

```
{
"ReturnValue": 1726,
"ReturnCode": "0"
}
```

2.6.7. CreateDQCRule

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to CreateDQCRule.
EntityId	Integer	Yes	The ID of the partition expression.
ProjectName	String	Yes	The name of the MaxCompute project.
Rules	String	Yes	The rules to be created.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	Boolean	Indicates whether the specified rules have been created.

Sample request

```
/? EntityId=4003922
&ProjectName=autotest
&Rules={
"addTemplateRules":[{
"property":"table_count",
"propertyType":"table",
"blockType":"0",
"templateId":"7",
"trend":"abs",
"warningThreshold":10,
"critical Threshold":50,
"ruleType":"0"
}],
"addSelfserviceRules":[{
"property":"table_count",
"propertyType":"table",
"blockType":"0",
"methodName":"count/table_count",
"w hereCondition":"id>100",
"checker":"9",
"trend":"abs",
"operator":">",
"expectValue":0,
"comment":"This is a rule.",
"ruleType":"1",
"warningThreshold":null,
"criticalThreshold":null}],
"envType":"o dps",
"projectName":"autotest",
"entityId":4003922}
&<Common request parameters>
```

Sample response

```
{
"ReturnValue": true,
"ReturnCode": "0"
}
```

2.6.8. CreateManualDag

You can call this API operation in OpenAPI Explorer for debugging. OpenAPI Explorer allows you to call API operations through its web interface. It automatically generates SDK code examples based on the specified request parameters.

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to CreateManualDag.
Bizdate	String	Yes	The data timestamp. Specify the date-based timestamp of data to be processed, such as 2019-04-03 00:00:00.
FlowName	String	Yes	The name of the manually triggered workflow.
ProjectName	String	Yes	The name of the DataWorks workspace.
DagPara	String	No	The workflow parameters in the format of `{"key1": "val1", "key2": "val2"}`.
NodePara	String	No	The node parameters in the format of Map<Long, String>. Long indicates the node ID, and String indicates a string of key-value pairs, such as `{"1232234255": "key1=val1 key2=val2", "321231412": "key1=val1 key2=val2"}`.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request.
ReturnCode	String	The return code.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	Long	The value of the dagId parameter.

Sample response

```
{
"RequestId": "****",
"ReturnCode": 0,
"ReturnMessage": "",
"ReturnErrorSolution": "",
"ReturnValue": 123142124124
}
```

2.6.9. CreateMetaSpiderJob

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to CreateMetaSpiderJob.
BaseId	String	Yes	The ID of the Apsara Stack account.
DatasourceId	Long	Yes	The ID of the data source.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
SpiderConfig	String	Yes	The configuration information about the job to create.
TenantId	Long	Yes	The ID of the tenant.

Parameter	Type	Required	Description
SpiderCron	String	No	The CRON expression of the job to create.

Response parameters

Parameter	Type
Data	Boolean
ErrCode	Long
ErrMsg	String
RequestId	String

Sample requests

```
http(s)://[Endpoint]/? Action=CreateMetaSpiderJob
&<Common request parameters>
```

2.6.10. CreateResGroup

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to CreateResGroup.
BaseId	String	Yes	The ID of the Apsara Stack account.
Name	String	Yes	The name of the resource group to create.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
ProjectIds	JSON	Yes	The IDs of the DataWorks workspaces.
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
Data	Object	The returned data.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=CreateResGroup
&BaseId=1486821399873474
&Name=test20190509
&ProjectId=75059
&ProjectIds=["75059"]
&TenantId=267883377178690
&<Common request parameters>
```

Sample success responses

```
{
"requestId": "0a98***e0b1b",
"data": {
"id": 334603176233473,
"identifier": "e86f****64dde44",
"name": "test1234",
"nick": null,
"type": "DI",
"mode": "ISOLATE",
"tenantId": 267883377178690,
"enableKp": false,
"cluster": "e86f1****64dde44",
"quota": null,
"isDefault": false,
"sequence": null,
"action": "create",
"specs": {
"resourceGroupTag": "1"
}
},
"errMsg": "success",
"errCode": 0
}
```


2.6.11. DeleteConnections

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to DeleteConnections.
ConnectionIds	String	Yes	The IDs of the data sources to delete.

Response parameters

Parameter	Type	Description

Parameter	Type	Description
Data	Boolean	Indicates whether the specified data resources have been deleted.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=DeleteConnections
&<Common request parameters>
```

2.6.12. DeleteDQCEntity

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to DeleteDQCEntity.
EntityId	Long	Yes	The ID of the partition filter expression.
EnvType	String	Yes	The type of the computing engine. Supported computing engines are MaxCompute, Hive, and Realtime Compute.
ProjectName	String	Yes	The name of the MaxCompute project.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	Boolean	Indicates whether the operation is successful.

Sample requests

```
/? Action=DeleteDQCEntity
&EntityId=4003922
&EnvType=odps
&ProjectName=autotest
&<Common request parameters>
```

Sample success responses

```
{
"ReturnValue": true,
"ReturnCode": "0"
}
```

2.6.13. DeleteDQCFollower

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to DeleteDQCFollower.
ProjectName	Long	No	The name of the MaxCompute project.
Id	Long	No	The ID of the subscription.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	Boolean	Indicates whether the operation is successful.

Sample request

```
/? Action=DeleteDQCFollower
&Id=1724
&ProjectName=autotest
&<Common request parameters>
```

Sample response

```
{
"ReturnValue": true,
"ReturnCode": "0"
}
```

2.6.14. DeleteDQCRule

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to DeleteDQCRule.
ProjectName	String	No	The name of the MaxCompute project.
Id	Long	No	The ID of the Data Quality check (DQC) rule to delete.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	Boolean	Indicates whether the specified rule has been deleted.

Sample requests

```
/? Action=DeleteDQCRule
&Id=279580661
&ProjectName=autotest
&<Common request parameters>
```

Sample success responses

```
{
"ReturnCode": "0",
"ReturnValue": true
}
```

2.6.15. DeleteProjectResGroup

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to DeleteProjectResGroup.
BaseId	String	Yes	The ID of the Apsara Stack account.
Identifier	String	Yes	The identifier of the resource group.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
ResGroupId	Long	Yes	The ID of the resource group.
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
Data	Boolean	Indicates whether the specified resource group has been deleted. Valid values: true and false. The value true indicates that the resource group has been deleted. The value false indicates that the resource group failed to be deleted.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=DeleteProjectResGroup
&BaseId=1486821399873474
&Identifier=e86f1a3ba2ca451eab8851ed564dde44
&ProjectId=75059 &ResGroupId=334603176233473
&TenantId=267883377178690
&<Common request parameters>
```

Sample success responses

```
{
"requestId": "0bc1746d15574010397142703e6afb",
"data": true,
"errMsg": "success",
"errCode": 0
}
```

2.6.16. DeleteResGroupGateWay

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to DeleteResGroupGateWay.
BaseId	String	Yes	The ID of the Apsara Stack account.
Identifier	String	Yes	The identifier of the resource group.
NodeName	String	Yes	The name of the gateway.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
Data	Boolean	Indicates whether the specified gateway has been deleted.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=DeleteResGroupGateWay
&BaseId=1486821399873474
&Identifier=3907f9a5832942f9b63b0908476b2506
&NodeName=hostname
&ProjectId=75059
&TenantId=267883377178690
&<Common request parameters>
```

Sample success responses

```
{
"requestId": "0bc1748b15574028120678613e5f16",
"data": true,
"errMsg": "success",
"errCode": 0
}
```

2.6.17. GetDagDetail

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetDagDetail.
AccountId	String	Yes	The ID of the account that is assigned by the scheduling system to call this operation.

Parameter	Type	Required	Description
AccountPwd	String	Yes	The password of the account that is assigned by the scheduling system to call this operation.
ClientName	String	Yes	The name of the client that is assigned by the scheduling system to call this operation.
BaseId	String	Yes	The ID of the Apsara Stack account that you use to call this operation.
ProjectEnv	String	Yes	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.
TenantId	Long	Yes	The ID of the tenant.
DagId	Long	No	The ID of the workflow instance.

Response parameters

Parameter	Type	Description
Count	Integer	The number of the returned workflow instances.
RequestId	String	The ID of the request.
ReturnCode	String	The return code.
ReturnMessage	String	The error message.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnValue	Long	The returned data.

Sample success responses

```
{
"returnCode": "0",
"returnErrorSolution": "",
"returnMessage": "",
"requestId": "14***4-e059-4bf4-a2c1-***323f",
"returnValue": [
{
"name": "Retroactive Instance Test 15",
"type": 3,
"rootNodeId": 1,
"status": 6,
"flowId": 1
}
],
"success": true
}
```


2.6.18. GetDataServiceApiDetail

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetDataServiceApiDetail.
ApiId	Long	Yes	The ID of the API to query.
Verbose	Long	No	Specifies whether to return the details of the API.

Response parameters

Parameter	Type	Description
Data	Boolean	The details of the DataService API.
ErrCode	Integer	The error code.
ErrMsg	String	The error message.

Parameter	Type	Description
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=GetDataServiceApiDetail
&<Common request parameters>
```

2.6.19. GetDataServiceAppDetail

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetDataServiceAppDetail.
BaseId	String	Yes	The ID of the Apsara Stack account.
ProjectId	Long	Yes	The ID of the DataWorks workspace.

Response parameters

Parameter	Type	Description
Data	Boolean	The details of the DataService application.
ErrCode	Integer	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=GetDataServiceAppDetail
&<Common request parameters>
```

2.6.20. GetDefaultTenant

Request parameters

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetDefaultTenant.

Response parameters

Parameter	Type	Description
ErrCode	Integer	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.
Tenant	String	The information about the tenant.
TenantCode	String	The code of the tenant.
TenantOwnerBaseId	String	The ID of the Apsara Stack account of the tenant owner.

Sample requests

```
http(s)://[Endpoint]/? Action=GetDefaultTenant
&<Common request parameters>
```

2.6.21. GetDQCEntity

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetDQCEntity.
ProjectName	String	No	The name of the DataWorks workspace.
TableName	String	No	The name of the table.

Parameter	Type	Required	Description
EnvType	String	No	The type of the computing engine. Supported computing engines are MaxCompute, Hive, and Realtime Compute.
MatchExpression	String	No	The partition filter expression. If you do not specify this parameter, the operation returns all the partition filter expressions of the specified table.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	String	The partition filter expressions that were queried.

Sample requests

```
http(s)://[Endpoint]/? Action=GetDQCEntity
&EnvType=odps
&MatchExpression=dt=${yyyymmdd-1}
&ProjectName=autotest
&TableName=test_dqc_decimal_1119_2
&<Common request parameters>
```

Sample success responses

```
{
"ReturnValue": {
"Entity": [
{
"EntityLevel": 0,
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-3}",
"Followers": "050624",
"OnDuty": "050624",
"Cal": 0
```

"Sql": 0,
"GmtCreate": "2018-11-26 15:06:32",
"EnvType": "odps", "Task": 0,
"HasRelativeNode": false, "Id": 4003918,
"TableName": "test_dqc_decimal_1119_2",
"GmtModify": "2018-11-26 15:06:32"
},
{
"EntityLevel": 0,
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Followers": "050624",
"OnDuty": "050624",
"Sql": 0,
"GmtCreate": "2018-11-26 22:31:13",
"EnvType": "odps",
"Task": 0,
"HasRelativeNode": false,
"Id": 4003922,
"TableName": "test_dqc_decimal_1119_2",
"GmtModify": "2018-11-26 22:31:13"
},
{
"EntityLevel": 0,
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-2}",
"Followers": "050624",
"OnDuty": "050624",
"Sql": 0,
"GmtCreate": "2018-11-26 23:18:34",
"EnvType": "odps",
"Task": 0,
"HasRelativeNode": false,
"Id": 4003923,
"TableName": "test_dqc_decimal_1119_2",
"GmtModify": "2018-11-26 23:18:34"
}
]
},
"ReturnCode": "0"
}

2.6.22. GetDQCFollower

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetDQCFollower.
ProjectName	String	No	The name of the DataWorks workspace.
EntityId	Long	No	The ID of the partition filter expression.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	String	The objects and applications that subscribe to the messages from Data Quality.

Sample requests

```
http(s)://[Endpoint]/? Action=GetDQCFollower
&EntityId=4003922
&ProjectName=odps
&<Common request parameters>
```

Sample success responses

```
{
"ReturnCode": "0",
"ReturnValue": {
"Follower": [
{
"ProjectName": "autotest",
"AlarmMode": 1,
"Follower": "050624",
"Id": 1726,
"TableName": "test_dqc_decimal_1119_2",
"EntityId": "4003922"
}
]
}
}
```


2.6.23. GetDQCRule

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetDQCRule.
ProjectName	String	No	The name of the DataWorks workspace.
EntityId	Long	No	The ID of the partition filter expression.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	List	The details of the rules.

Sample requests

```
http(s)://[Endpoint]/? Action=GetDQCRule
&EntityId=4003922
&ProjectName=autotest
&<Common request parameters>
```

Sample success responses


```
{
"ReturnValue": {
"TemplateRules": {
"TemplateRule": [
{
"TemplateName": "SQL task table rows, 1,7, 30 days fluctuation test",
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"WarningThreshold": "40",
"RuleCheckerRelationId": 1008005,
"RuleType": 0,
"MethodId": 8,
"CriticalThreshold": "80",
"FixCheck": false,
"OnDuty": "050624",
"Property": "table_count",
"TemplateId": 7,
"PropertyKey": "table_count",
"MethodName": "table_count",
"HistoryWarningThreshold": "history max:40%,history min:10%",
"BlockType": 1,
"CheckerId": 7,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580663,
"EntityId": 4003922,
"HistoryCriticalThreshold": "history max:80%,history min:50%",
"Trend": "abs"
},
{
"TemplateName": "SQL task table rows, 1,7, 30 days fluctuation test",
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"WarningThreshold": "10",
"RuleCheckerRelationId": 1008007
```

```
"RuleCheckerRelationId": 1008007,
"RuleType": 0,
"MethodId": 8,
"CriticalThreshold": "50",
"FixCheck": false,
"OnDuty": "050624",
"Property": "table_count",
"TemplateId": 7,
"PropertyKey": "table_count",
"MethodName": "table_count",
"HistoryWarningThreshold": "history max:10%,history min:10%",
"BlockType": 0,
"CheckerId": 7,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580665,
"EntityId": 4003922,
"HistoryCriticalThreshold": "history max:50%,history min:50%",
"Trend": "abs"
},
{
"TemplateName": "SQL task table rows, 1,7, 30 days fluctuation test",
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"WarningThreshold": "10",
"RuleCheckerRelationId": 1008009,
"RuleType": 0,
"MethodId": 8,
"CriticalThreshold": "50",
"FixCheck": false,
"OnDuty": "050624",
"Property": "table_count",
"TemplateId": 7,
"PropertyKey": "table_count",
"MethodName": "table_count",
"HistoryWarningThreshold": "history max:10%,history min:10%",
"BlockType": 0,
"CheckerId": 7,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580667,
"EntityId": 4003922,
"HistoryCriticalThreshold": "history max:50%,history min:50%",
```

```
"Trend": "abs"
}
]
},
"SelfserviceRules": {
"SelfserviceRule": [
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,
"RuleCheckerRelationId": 1008004,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",
"PropertyKey": "table_count",
"MethodName": "count/table_count",
"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580662,
"CheckerId": 6,
"EntityId": 4003922,
"Trend": "abs"
},
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,
"RuleCheckerRelationId": 1008006,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",
"PropertyKey": "table_count",
"MethodName": "count/table_count",
```

"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580664,
"CheckerId": 6,
"EntityId": 4003922,
"Trend": "abs"
},
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,
"RuleCheckerRelationId": 1008008,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",
"PropertyKey": "table_count",
"MethodName": "count/table_count",
"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580666,
"CheckerId": 6,
"EntityId": 4003922,
"Trend": "abs"
},
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,
"RuleCheckerRelationId": 1008010,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",

```
"PropertyKey": "table_count",
"MethodName": "count/table_count",
"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580668,
"CheckerId": 6,
"EntityId": 4003922, "Trend": "abs"
}
]
}
},
"ReturnCode": "0"
}
```


2.6.24. GetNodeDetail

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetNodeDetail.
AccountId	String	Yes	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	Yes	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	Yes	The ID of the Apsara Stack account that you use to call this operation.
ClientName	String	Yes	The name of the client that is assigned by the scheduling system to call this operation.
NodeId	Long	Yes	The ID of the node to query.

Parameter	Type	Required	Description
ProjectEnv	String	Yes	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.
TenantId	Long	Yes	The ID of the tenant.
IsDetail	Boolean	No	Specifies whether to return the detailed information about this node. Valid values: true and false.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	String	None

Sample requests

```
http(s)://[Endpoint]/? Action=GetNodeDetail
&EntityId=4003922
&ProjectName=autotest
&<Common request parameters>
```

Sample success responses

```
{
"ReturnValue": {
"TemplateRules": {
```

```
"TemplateRule": [
{
"TemplateName": "SQL task table rows, 1,7, 30 days fluctuation test",
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"WarningThreshold": "40",
"RuleCheckerRelationId": 1008005,
"RuleType": 0,
"MethodId": 8,
"CriticalThreshold": "80",
"FixCheck": false,
"OnDuty": "050624",
"Property": "table_count",
"TemplateId": 7,
"PropertyKey": "table_count",
"MethodName": "table_count",
"HistoryWarningThreshold": "history max:40%,history min:10%",
"BlockType": 1,
"CheckerId": 7,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580663,
"EntityId": 4003922,
"HistoryCriticalThreshold": "history max:80%,history min:50%",
"Trend": "abs"
},
{
"TemplateName": "SQL task table rows, 1,7, 30 days fluctuation test",
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"WarningThreshold": "10",
"RuleCheckerRelationId": 1008007,
"RuleType": 0,
"MethodId": 8,
"CriticalThreshold": "50",
"FixCheck": false,
"OnDuty": "050624",
"Property": "table_count",
"TemplateId": 7,
"PropertyKey": "table_count",
"MethodName": "table_count",
```

"HistoryWarningThreshold": "history max:10%,history min:10%",
"BlockType": 0,
"CheckerId": 7,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580665,
"EntityId": 4003922,
"HistoryCriticalThreshold": "history max:50%,history min:50%",
"Trend": "abs"
},
{
"TemplateName": "SQL task table rows, 1,7, 30 days fluctuation test",
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"WarningThreshold": "10",
"RuleCheckerRelationId": 1008009,
"RuleType": 0,
"MethodId": 8,
"CriticalThreshold": "50",
"FixCheck": false,
"OnDuty": "050624",
"Property": "table_count",
"TemplateId": 7,
"PropertyKey": "table_count",
"MethodName": "table_count",
"HistoryWarningThreshold": "history max:10%,history min:10%",
"BlockType": 0,
"CheckerId": 7,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580667,
"EntityId": 4003922,
"HistoryCriticalThreshold": "history max:50%,history min:50%",
"Trend": "abs"
}
]
},
"SelfserviceRules": {
"SelfserviceRule": [
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,

"RuleCheckerRelationId": 1008004,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",
"PropertyKey": "table_count",
"MethodName": "count/table_count",
"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580662,
"CheckerId": 6,
"EntityId": 4003922,
"Trend": "abs"
},
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,
"RuleCheckerRelationId": 1008006,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",
"PropertyKey": "table_count",
"MethodName": "count/table_count",
"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580664,
"CheckerId": 6,
"EntityId": 4003922,
"Trend": "abs"
},
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",

"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,
"RuleCheckerRelationId": 1008008,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",
"PropertyKey": "table_count",
"MethodName": "count/table_count",
"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580666,
"CheckerId": 6,
"EntityId": 4003922,
"Trend": "abs"
},
{
"ProjectName": "autotest",
"MatchExpression": "dt=${yyyymmdd-1}",
"Checker": 9,
"RuleCheckerRelationId": 1008010,
"WhereCondition": "id>100",
"RuleType": 1,
"MethodId": 21,
"FixCheck": true,
"CheckerName": "compared with a fixed value",
"OnDuty": "050624",
"Property": "table_count",
"PropertyKey": "table_count",
"MethodName": "count/table_count",
"BlockType": 0,
"TableName": "test_dqc_decimal_1119_2",
"Id": 279580668,
"CheckerId": 6,
"EntityId": 4003922, "Trend": "abs"
}
]
}

```
},
"ReturnCode": "0"
}
```

2.6.25. GetNodeUpdateStatus

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetNodeUpdateStatus.
AccountId	String	No	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	No	The password of the account that is assigned by the scheduling system to call this operation.
ClientName	String	No	The name of the client that is assigned by the scheduling system to call this operation.
BaseId	String	No	The ID of the Apsara Stack account that you use to call this operation.
RequestId	String	No	The ID of the request. You can use this ID to call the ModifyNode operation to modify a node. The ModifyNode operation then returns this ID as the value of the ReturnValue response parameter.
TenantId	LONG	No	The ID of the tenant.

Parameter	Type	Required	Description
IsDetail	Boolean	No	Specifies whether to return the detailed information about the node update. Valid values: true and false.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	String	The value of the RequestId parameter.

Sample success responses


```
{
"returnCode": "0",
"returnErrorSolution": "",
"returnMessage": "",
"requestId": "0bbaa2451557469839878 6934e12b8",
"returnValue": {
"requestId": "0a64886b-1d63-4e5c-ad3e-d4eda4c2381c",
"createTime": 1555065345000,
"name": "ide-base",
"status": "SUCCESS",
"isDetail": true,
"flowId": 28620441,
"executeDetails": [
{
"uniqueId": "9745681_2",
"cloudUUID": 111164284,
"executeStatus": "SUCCESS",
"executeMsg": "success"
```

```
"executeMsg": "success",
"dagId": null
},
{
"uniqueId": "9745682_5",
"cloudUUID": 111001035,
"executeStatus": "SUCCESS",
"executeMsg": "success",
"dagId": null
},
{
"uniqueId": "9745683_9",
"cloudUUID": 111087244,
"executeStatus": "SUCCESS",
"executeMsg": "success",
"dagId": null
},
{
"uniqueId": "9745680_12",
"cloudUUID": 111087242,
"executeStatus": "SUCCESS",
"executeMsg": "success",
"dagId": null
},
{
"uniqueId": "9745680_12",
"cloudUUID": 111087242,
"executeStatus": "INIT",
"executeMsg": null,
"dagId": null
}
]
},
"success": true
}
```


2.6.26. GetResGroupAk

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to GetResGroupAk.
BaseId	String	Yes	The ID of the Apsara Stack tenant account.
Identifier	String	Yes	The identifier of the resource group.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample request

```
http(s)://[Endpoint]/? Action=GetResGroupAk
&BaseId=148****873474
&Identifier=3907f9****6b2506
&ProjectId=75059
&TenantId=2678***78690
&<Common request parameters>
```

Sample response

```
{
"requestId": "0a98a368***97e56fd",
"data": {
"username": "zz_3907f9***08476b2506",
"password": "vcn9******",
"rank": 4,
"ossBackUp": false,
"callback": false,
"callbackUrl": ""
},
"errMsg": "success",
"errCode": 0
}
```


2.6.27. GetResGroupGatewayList

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to GetResGroupGatewayList.
BaseId	String	Yes	The ID of the Apsara Stack account.
Identifier	String	Yes	The identifier of the resource group.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
Data	List	The returned data.
ErrCode	Long	The error code.
ErrMsg	String	The error message.

Parameter	Type	Description
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=GetResGroupGatewayList
&BaseId=1486****73474
&Identifier=3907f9a5******76b2506
&ProjectId=75059
&TenantId=2678****78690
&<Common request parameters>
```

Sample success responses

```
{
"requestId": "0a98a3541557401599643271 9e2c97",
"data": [{
  "clusterName": "3907f9a5832942f9b63b0908476b2506",
  "nodeName": "1.2.3.50",
  "nodeAddress": "1.2.3.50",
  "maxSlot": 32,
  "useSlot": 0,
  "state": 1,
  "live": false,
  "startMon": false,
  "regionId": 1,
  "nodeProperties": "{\"memory\":8,\"vCpu\":4}",
  "vCpu": 4,
  "memory": 8,
  "useDmu": 0
  }],
  "errMsg": "success",
  "errCode": 0
}
```

2.6.28. GetResGroupList

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetResGroupList.
AlisaDetail	Boolean	Yes	Indicates whether to return the node details.
BaseId	String	Yes	The ID of the Apsara Stack account.
Level	String	Yes	The level of the resource group. Valid values: project and tenant. The value project indicates that the resource group belongs to a project. The value tenant indicates that the resource group belongs to a tenant.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
ResourceGroupType	String	Yes	The type of the resource group.
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
Data	List	The returned data.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=GetResGroupList
&AlisaDetail=true
&BaseId=14*****3474
&Level=project
&ProjectId=75059
&ResourceGroupType=di
&TenantId=2678*****690
&<Common request parameters>
```

Sample success responses


```
{
"requestId": "0a98a3****52214e2c97",
"data": [{
"id": 29810****9522,
"identifier": "group_267****178690",
"name": "default resource group",
"nick": null,
"isDefault": true,
"networkType": null,
"nodes": null,
"maxSlot": null,
"useSlot": null,
"useDmu": null,
"maxDmu": null,
"projects": null,
"resourceGroupType": null,
"diResourceGroupType": 0,
"buyType": "pay-as-you-go"
},
{
"id": 200000003001,
"identifier": "3907f9a58***08476b2506",
"name": "y1y1y1",
"nick": null,
"isDefault": false,
"networkType": 1,
"nodes": [{
"clusterName": "3907f9a583****8476b2506",
"nodeName": "1.2.3.50",
"nodeAddress": "1.2.3.50"
```

```
"nodeAddress": "1.2.3.50",
"maxSlot": 32,
"useSlot": 0,
"state": 1,
"live": false,
"startMon": false,
"regionId": 1,
"nodeProperties": "{\"memory\":8,\"vCpu\":4}",
"vCpu": 4,
"memory": 8,
"useDmu": 0
}],
"maxSlot": 32,
"useSlot": 0,
"useDmu": 0,
"maxDmu": 16,
"projects": null,
"resourceGroupType": null,
"diResourceGroupType": 1,
"buyType": "pay-as-you-go"
},
{
"id": 33224***49920,
"identifier": "c43d****cb7fef18c137",
"name": "cfr",
"nick": null,
"isDefault": false,
"networkType": 1,
"nodes": [{
"clusterName": "c43da72697****c137",
"nodeName": "deide",
"nodeAddress": "10.0.0.22",
"maxSlot": 8,
"useSlot": 0,
"state": 1,
"live": false,
"startMon": false,
"regionId": 1,
"nodeProperties": "{\"memory\":4,\"vCpu\":2}",
"vCpu": 2,
"memory": 4,
```

```
"useDmu": 0
}],
"maxSlot": 8,
"useSlot": 0,
"useDmu": 0,
"maxDmu": 4,
"projects": null,
"resourceGroupType": null,
"diResourceGroupType": 1,
"buyType": "pay-as-you-go"
},
{
"id": 329080157014786,
"identifier": "bd49c8014fb64f53835793e94932a9ac",
"name": "hello",
"nick": null,
"isDefault": false,
"networkType": 1,
"nodes": [{
"clusterName": "bd49c801******e94932a9ac",
"nodeName": "djed***qduew",
"nodeAddress": "0.1.0.0",
"maxSlot": 24,
"useSlot": 0,
"state": 1,
"live": false, "startMon": false, "regionId": 1,
"nodeProperties": "{\"memory\":8,\"vCpu\":8}", "vCpu": 8,
"memory": 8,
"useDmu": 0
}],
"maxSlot": 24,
"useSlot": 0,
"useDmu": 0,
"maxDmu": 12,
"projects": null,
"resourceGroupType": null,
"diResourceGroupType": 1,
"buyType": "pay-as-you-go"
}],
"errMsg": "success",
"errCode": 0
```

```
}
```

2.6.29. GetTableColumn

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to GetTableColumn.
BaseId	String	Yes	The ID of the Apsara Stack account.
DatasourceName	String	Yes	The name of the data source.
DatasourceType	String	Yes	The type of the data source.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
SubType	String	Yes	The subtype of the data source. Currently, only MySQL is supported.
Table	String	Yes	The name of the table in the data source.
TenantId	Long	Yes	The ID of the tenant.
Id	String	No	The ID of the data source.

Response parameters

Parameter	Type	Description
Data	-	The fields of the specified table in the data source.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=GetTableColumn
&BaseId=148682***73474
&DatasourceName=odps_first
&DatasourceType=odps
&ProjectId=75059
&SubType=public
&Table=t
&TenantId=2678***78690
&<Common request parameters>
```

Sample success responses

```
{
"requestId": "0bc1748b15574030794246845e5f16",
"data": {
"splitPk": [],
"columns": [{
"name": "id",
"type": "STRING"
}],
"partitionColumns": [],
"extraInfo": {}
},
"errMsg": "success",
"errCode": 0
}
```

2.6.30. GetTableList

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to GetTableList.
BaseId	String	Yes	The ID of the Apsara Stack tenant account.

Parameter	Type	Required	Description
DatasourceName	String	Yes	The name of the data store.
DatasourceType	String	Yes	The type of the data store.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
SubType	String	Yes	The subtype of the data store.
Table	String	Yes	The name of the table in the data store.
TenantId	Long	Yes	The ID of the tenant.
Id	Long	No	The ID of the data store.
PageSize	Long	No	The number of entries to return on each page.

Response parameters

Parameter	Type	Description
Data	-	The returned data.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample request

```
http(s)://[Endpoint]/? Action=GetTableList
&BaseId=1486821399873474
&DatasourceName=odps_first
&DatasourceType=odps
&ProjectId=75059
&SubType=public
&Table=t
&TenantId=267883377178690
&<Common request parameters>
```

Sample response


```
{
"requestId": "0bc17****2e5f16",
"data": {
"tables": ["c", "t", "t1", "t2", "wowo", "zzyx", "oss_test", "oss_tes00t", "test_input"]
},
"errMsg": "success",
"errCode": 0
}
```


2.6.31. GetTaskLog

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to GetTaskLog.
AccountId	String	Yes	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	Yes	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	Yes	The ID of the Apsara Stack account that you use to call this operation.
ClientName	String	Yes	The name of the client that is assigned by the scheduling system to call this operation.

Parameter	Type	Required	Description
ProjectEnv	String	Yes	The type of the environment. Valid values: PROD and DEV. The value PROD specifies to return the information about the node instances in the production environment that meet the search conditions. The value DEV specifies to return the information about the node instances in the development environment that meet the search conditions.
TenantId	Long	Yes	The ID of the tenant. Set the value to 1.
TaskId	Long	No	The ID of the node instance.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	None
ReturnMessage	String	The error message.
ReturnValue	String	The logs that were returned.

Sample requests

```
http(s)://[Endpoint]/? Action=GetTaskLog
&AccountId=C00000
&AccountPwd=account_pwd
&BaseId=0I0001
&ClientName=aone_app_name
&ProjectEnv=DEV
&TenantId=1
&TaskId=20000000
&<Common request parameters>
```

Sample success responses

```
{
"ReturnValue": "I'm alog",
"ReturnCode": "0",
"RequestId": "023dbca2-3****f23fa26",
"ReturnErrorSolution": "",
"ReturnMessage": ""
}
```

2.6.32. GetUserInfo

Request parameters

Parameter	Type	Required	Description
Action	STRING	Yes	The operation that you want to perform. Set the value to *GetUserInfo*.

Response parameters

Parameter	Type	Description
ErrCode	INTEGER	The returned error code.
ErrMsg	STRING	The returned message.
RequestId	STRING	The ID of the request.
User	N/A	The information about the user that was queried.

2.6.33. ListConnection

You can call this operation to query data sources.

Request parameters

Parameter	Type	Required	Example	Description
ProjectId	Long	Yes	12345	The ID of the data source.
Name	String	No	"myds"	The name of the data source. This parameter must be specified when you want to query the AccessKey of a computing engine. By default, do not set this parameter.
Keyword	Boolean	No	"mydatasource"	Specifies whether to hide the specified sensitive fields. Default value: *false*.
Status	Integer	No	0	The status of the data source. Valid values: 0, 1, and 2. • 0: The data source is deleted. • 1: The data source is normal. • 2: The data source is disabled.
BindEngineId	Long	No	12345	The ID of the compute engine that is bound with the data source.

Parameter	Type	Required	Example	Description
EnvType	Integer	No	1	The type of the environment. Valid values: 0 and 1. • 0: the development environment. • 1: the production environment.
BaseId	String	No	"034302"	The ID of the Apsara Stack account. This parameter must be specified when you want to query the AccessKey of a computing engine. By default, do not set this parameter.
Verbose	Boolean	No	true	Specifies whether to return the details about the connection string of the data source. Default value: false.
PageNum	Integer	No	1	The number of the page to return. Default value: 1.
PageSize	Integer	No	10	The number of entries to return on each page. Default value: 10.

Response parameters

Parameter	Type	Example	Description

Parameter	Type	Example	Description
requestId	String	c9afd89b-7aef-47c9-9902-e0500d843268	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting. The request ID must be globally unique.
errCode	Long	0	The error code. • The value 0 indicates that the operation is successful. • A value greater than 0 indicates the error code of the error that occurred.
errMsg	String	success	The error message. If the value of the errCode parameter is 0, the value of the errMsg parameter is a null string or success.
data	PagingResult<Connection>	N/A	The returned data.
TotalNum	Integer	N/A	The total number of entries that were returned.
PageNum	Integer	N/A	The page number of the returned page.
PageSize	Integer	N/A	The number of entries returned per page.
Connections	List<Connection>	N/A	The list of data sources that were returned.
Id	Long	12345	The ID of the data source.
TenantId	Long	12345	The ID of the tenant.
ProjectId	Long	12345	The ID of the DataWorks workspace.
Name	String	"testDataSource"	The name of the data source.

Parameter	Type	Example	Description
Type	String	"mysql"	The type of the connection string.
SubType	String	""	The subtype of the connection string.
EnvType	Integer	0	The type of the environment. Valid values: 0 and 1. The value 0 indicates the development environment. The value 1 indicates the production environment.
Connection	String	{"username":"u1","password":"p1"}	The connection string of the data source.
Sequence	Integer	300	The sequence number of the data source.
Status	Integer	0	The status of the data source. Valid values: 0, 1, and 2. • 0: The data source is deleted. • 1: The data source is normal. • 2: The data source is disabled.
Description	String	"this is a testdatasource"	The description of the data source.
Shared	Boolean	true	Indicates whether the data source is shared among users of the tenant.
GmtCreate	Date	2018-11-11 00:00:00	The time when the data source was created.
GmtModified	Date	2018-11-11 00:00:00	The time when the data source was last modified.
Operator	String	"032214"	The user who modified the data source last time.

Parameter	Type	Example	Description
ConnectTime	Date	2018-11-11 00:00:00	The time when the data source was last connected.
ConnectStatus	Integer	0	The connection status of the data source. Valid values: 0, 1, and 2. • 0: The connectivity test has not been run. • 1: The data source is successfully connected. • 2: The data source failed the connectivity test.

Sample requests

```
/? ProjectId=12345
&Keyword="mydatasource"
&<Common request parameters>
```

Sample success responses

JSON format

```
{
 "data":{
  "TotalNum":35,
  "PageNum":1,
  "PageSize":10,
  "Connections":[
   {
    "Id":12345,
    "TenantId":12345,
    "ProjectId":12345,
    "Name":"testDataSource",
    "Type":"mysql",
    "SubType":"",
    "EnvType":0,
    "Connection":"{\"username\":\"u1\",\"password\":\"p1\"}",
    "Sequence":300,
    "Status":0,
    "Description":"this is a test datasource",
    "Shared":0,
    "GmtCreate":"2018-11-11 00:00:00",
    "GmtModified":"2018-11-11 00:00:00",
    "Operator":"032214",
    "ConnectTime":"2018-11-11 00:00:00",
    "ConnectStatus":0
   },...
  ]
 },
 "errCode":0,
 "errMsg":"success",
 "requestId":"c9afd89b-7aef-47c9-9902-e0500d843268"
}
```


Sample error responses

JSON format

```
{
 "data":null,
 "errCode":1101002001,
 "errMsg":"invalid parameter",
 "requestId":"c9afd89b-7aef-47c9-9902-e0500d843268"
}
```

2.6.34. ListDataServiceApps

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListDataServiceApps.
BaseId	String	No	The ID of the Apsara Stack account.
AppName	String	No	The name of the application to query.
PageSize	Integer	No	The number of entries to return on each page.
PageNum	Integer	No	The number of the page to return.

Response parameters

Parameter	Type	Description
Data	N/A	The returned data.
ErrCode	Integer	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

2.6.35. ListDataServiceAuthedApi

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListDataServiceAuthedApi.
BaseId	String	Yes	The ID of the Apsara Stack account.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
ApiStatus	String	No	The status of the operation to query.
PageNum	Integer	No	The number of the page to return.
PageSize	Integer	No	The number of entries to return on each page.
Verbose	Boolean	No	Indicates whether to return the details of the operation.

Response parameters

Parameter	Type	Description
Data	N/A	The returned data.
ErrCode	Integer	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

2.6.36. ListPermission

You can call this operation to query permissions.

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListPermission.

Parameter	Type	Required	Description
baseId	String	No	The ID of the Apsara Stack account.
tenantId	Long	No	The ID of the tenant.
projectId	Long	No	The ID of the DataWorks workspace.
modules	String	No	The modules of the DataWorks workspace.

Response parameters

Parameter	Type	Description
Data	N/A	The returned data.
errCode	Integer	The error code. • The value 0 indicates that the operation is successful. • A value greater than 0 indicates the error code of the error that occurred.
requestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting. The request ID must be globally unique.

Sample requests

```
/? Action=ListPermission
&baseId=
&modules=
&projectId=
&tenantId=
&<Common request parameters>
```

2.6.37. ListProjectModule

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListProjectModule.
ProjectId	Long	No	The ID of the DataWorks workspace.

Response parameters

Parameter	Type	Description
ErrCode	Integer	The error code.
ErrMsg	String	The error message.
Modules	N/A	The modules of the DataWorks workspace.
RequestId	String	The ID of the request.

2.6.38. ListProjectModules

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListProjectModules.
ProjectId	Long	No	The ID of the DataWorks workspace.

Response parameters

Parameter	Type	Description
ErrCode	Integer	The error code.
ErrMsg	String	The error message.
Modules	N/A	The modules of the DataWorks workspace.
RequestId	String	The ID of the request.

2.6.39. ListProject

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListProject.

Response parameters

Parameter	Type	Description
ErrCode	Long	The error code.
ErrMsg	String	The error message.
ProjectList	N/A	The DataWorks workspaces that were queried.
RequestId	String	The ID of the request.

2.6.40. ListTenantModule

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListTenantModule.
TenantId	Long	No	The ID of the tenant.

Response parameters

Parameter	Type	Description
ErrCode	Integer	The error code.
ErrMsg	String	The error message.
Modules	N/A	The modules of the tenant.
RequestId	String	The ID of the request.

2.6.41. ListUserPermission

You can call this operation to query permissions of a user.

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ListUserPermission.
BaseId	String	No	The ID of the Apsara Stack account.
TenantId	Long	No	The ID of the tenant.
ProjectId	Long	No	The ID of the DataWorks workspace.
Modules	String	No	The modules of the DataWorks workspace.

Response parameters

Parameter	Type	Description
Data	N/A	The returned data.
ErrCode	Integer	The error code. • The value 0 indicates that the operation is successful. • A value greater than 0 indicates the error code of the error that occurred.
ErrMsg	String	The error message. If the value of the errCode parameter is 0, the value of the errMsg parameter is a null string or success.
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting. The request ID must be globally unique.

2.6.42. ModifyBusiness

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ModifyBusiness.
AccountId	String	No	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	No	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	No	The ID of the Apsara Stack account that you use to call this operation.

BusinessModifyDto	String	No	{
			"executeMethod": "UPDATE_BUSINESS",
			"bizId": 100000003,
			"bizKey": "biz_key_01",
			"bizName": "biz_name_01_xxxx",
			"solId": 100000002,
			"appId": 78918,
			"tenantId": 2797***8706
			}

ClientName	String	No	The name of the client that is assigned by the scheduling system to call this operation.
TenantId	Long	No	The ID of the tenant.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	None
ReturnMessage	String	The error message.
ReturnValue	String	The information about the workflow that was modified.

2.6.43. ModifyNode

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ModifyNode.
AccountId	String	No	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	No	The password of the account that is assigned by the scheduling system to call this operation.
ClientName	String	No	The name of the client that is assigned by the scheduling system to call this operation.

Parameter	Type	Required	Description

ModifyDtoList	String	No	[{
			"executeMethod": "CREATE_NODE",
			"nodeType": 0,
			"cronExpress": "0 0 0 * * ?",
			"owner": "{{user_id}}",
			"createUser": "{{user_id}}",
			"modifyUser": "{{user_id}}",
			"dependentType": 0,
			"nodeName": "locale_test_node118",
			"prgType": 99,
			"priority": 0,
			"appId": 14255,
			"cycType": 1,
			"codeSrc": "",
			"nodeFrom": "postman",
			"inputs": [{
			"data": "test_dev_prod_depend_single_root"
			}],
			"outputs": ["data": "locale_test_node118out" }],
			"uniqueId": "locale_test_node6"
			}]

Parameter	Type	Required	Description
ProjectEnv	String	No	The type of the environment. Valid values: DEV and PROD. The value DEV specifies the development environment. The value PROD specifies the production environment.
BaseId	String	No	The ID of the Apsara Stack account that you use to call this operation.
TenantId	Long	No	The ID of the tenant.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	String	The value of the RequestId parameter. The request ID is returned after you send an asynchronous request. You can use this ID to call the GetNodeUpdateStatus operation to query the node status.

Sample success responses

```
{
"returnCode": "0",
"returnErrorSolution": "",
"returnMessage": "",
"requestId": "82e4c****a39fbda",
"returnValue": "82e4c04****f5a39fbda",
"success": true
}
```


2.6.44. ModifySolution

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ModifySolution.
AccountId	String	No	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	No	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	No	The ID of the Apsara Stack account that you use to call this operation.
ClientName	String	No	The name of the client that is assigned by the scheduling system to call this operation.
TenantId	Long	No	The ID of the tenant.
SolutionModifyDto	String	No	The information about the solution in the JSON format.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	String	None

Example of the SolutionModifyDto parameter

```
{
"executeMethod": "UPDATE_SOLUTION",
"solId": 100000001,
"solName": "sol_name_01_xxxx",
"appId": 78918,
"tenantId": 279717776488706
}
```

Sample success responses

```
{
"returnCode": "0",
"returnErrorSolution": "",
"returnMessage": "",
"requestId": "53ca5f***11bab4d0975",
"returnValue": {
"solId": 100000001,
"solName": "sol_name_01_xxxx",
"appId": 78918,
"tenantId": 279717776488706
},
"success": true
}
```

2.6.45. RerunTask

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to RerunTask.
AccountId	String	Yes	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	Yes	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	Yes	The ID of the Apsara Stack account that you use to call this operation.
ClientName	String	Yes	The name of the client that is assigned by the scheduling system to call this operation.
ProjectEnv	String	Yes	The type of the environment.
TaskId	Long	Yes	None
TenantId	Long	Yes	The ID of the tenant.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request.
ReturnCode	String	The error code.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	Boolean	Indicates whether the node is successfully rerun.

Sample success responses

```
{
"RequestID": "",
"RetrunCode": 12344,
"ReturnErrorSolution": "****",
"ReturnMessage": "****",
"ReturnValue": true
}
```


2.6.46. ResumeTask

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to ResumeTask.
AccountId	String	No	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	No	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	No	The ID of the Apsara Stack account that you use to call this operation.
ClientName	String	No	The name of the client that is assigned by the scheduling system to call this operation.
ProjectEnv	String	No	The type of the environment. Valid values: DEV and PROD.
TaskId	Long	No	The ID of the node to resume.
TenantId	Long	No	The ID of the tenant.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The error code.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.

Sample success responses

```
{
"RequestId": "1212adf***s-adfsdsf",
"ReturnCode": 0,
"ReturnMessage": "",
"ReturnErrorSolution": "",
"ReturnValue": true
}
```

2.6.47. SearchBusiness

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to SearchBusiness.
AccountId	String	No	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	No	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	No	The ID of the Apsara Stack account that you use to call this operation.

Parameter	Type	Required	Description
BizId	Long	No	Optional. The ID of the workflow.
ClientName	String	No	The name of the client that is assigned by the scheduling system to call this operation.
BizName	String	No	The name of the workflow.
BizKey	String	No	Optional. The key of the workflow.
TenantId	Long	No	The ID of the tenant.
AppId	Long	No	The ID of the DataWorks workspace.
AppIds	String	No	Optional. The IDs of the DataWorks workspaces. Separate multiple IDs with commas (,).

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue	-	None

Sample success responses

```
{
"returnCode": "0",
"returnErrorSolution": "",
"returnMessage": "",
"requestId": "d4605f***145312",
"returnValue": [
{
"bizId": 10010759,
"bizName": "Control Node 0",
"bizKey": "10010759",
"appId": 12344,
"tenantId": 121**212
}
],
"success": true
}
```


2.6.48. SearchManualDagNodeInstance

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to SearchManualDagNodeInstance.
ProjectName	String	No	The name of the DataWorks workspace.
DagId	Long	No	The ID of the instance for a manually triggered workflow. You can obtain this ID by calling the CreateManualDag operation.

Response parameters

Parameter	Type	Description
Data	String	The instances of the manually triggered workflow.

Parameter	Type	Description
ErrCode	String	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.
Success	Boolean	Indicates whether the operation is successful.

2.6.49. SearchSolution

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to SearchSolution.
AccountId	String	No	The ID of the account that is assigned by the scheduling system to make the API request.
AccountPwd	String	No	The password of the account that is assigned by the scheduling system to make the API request.
BaseId	String	No	The ID of the Apsara Stack tenant account that you use to call this operation.
ClientName	String	No	The name of the client that is assigned by the scheduling system to make the API request.
SolId	Long	No	(Optional) The ID of the solution.
SolName	String	No	The name of the solution.
TenantId	Long	No	The ID of the tenant.
AppId	Long	No	The ID of the DataWorks workspace.

Response parameters

Parameter	Type	Description
RequestId	String	The ID of the request. If an error occurs, you can send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The error code.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.

Sample response

```
{
"returnCode": "0",
"returnErrorSolution": "",
"returnMessage": "",
"requestId": "1381****b98d87c",
"returnValue": [
{
"solId": 100000007,
"solName": "Solution Test 222",
"appId": 33646,
"tenantId": 27***9020672
}
],
"success": true
}
```

2.6.50. SearchTasks

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to SearchTasks.

Parameter	Type	Required	Description
AccountId	String	Yes	The ID of the account that is assigned by the scheduling system to call this operation.
AccountPwd	String	Yes	The password of the account that is assigned by the scheduling system to call this operation.
BaseId	String	Yes	The ID of the Apsara Stack account that you use to call this operation.
ClientName	String	Yes	The name of the client that is assigned by the scheduling system to call this operation.
ProjectEnv	String	Yes	The type of the environment. Valid values: PROD and DEV. The value PROD specifies to return the information about the node instances in the production environment that meet the search conditions. The value DEV specifies to return the information about the node instances in the development environment that meet the search conditions.
NodeId	Long	No	The ID of the node. You can view the ID on the node configuration page in the DataWorks console.
TenantId	Long	Yes	The ID of the tenant. Set the value to 1.

Parameter	Type	Required	Description
DagId	Long	No	The ID of the workflow instance to which the node instances belong. When you call the CreateDag operation to create a workflow instance, the ID of the workflow instance is returned. You can use this ID to query the workflow instance.
Bizdate	String	No	The date-based timestamp. The value is in the format of yyyy-mm-dd 00:00:00. You can use this data timestamp to filter the node instances.

Response parameters

Parameter	Type	Description
Count	Integer	The number of the node instances that meet the search conditions.
RequestId	String	The ID of the request. If an error occurs, send the request ID to the developers of the scheduling system for troubleshooting.
ReturnCode	String	The return code. The value 0 indicates that the operation is successful.
ReturnErrorSolution	String	The solution to resolve the issue.
ReturnMessage	String	The error message.
ReturnValue		None

Sample requests

```
http(s)://[Endpoint]/? Action=SearchTasks
&AccountId=C00000
&AccountPwd=account-pwd
&BaseId=0I0001
&ClientName=aone_app_name
&ProjectEnv=PROD &TenantId=1
&Bizdate=2018-11-17 00:00:00
&DagId=10000000
&NodeId=1000001
&<Common request parameters>
```

Sample success responses

```
{
"Count": 3236869,
"ReturnValue": [
{
"DqcInstance": "",
"BeginWaitResTime": "",
"ResGroupId": 30004716,
"AppId": 14255,
"TaskId": 9993181585,
"PrgName": "",
"Gmtdate": "2018-11-17 00:00:00",
"NodeId": 20636213,
"DqcDescription": "",
"BeginRunningTime": "",
"AlisaReturnTime": "",
"FinishTime": "",
"GwLogLocalFile": "",
"DagId": 300041822,
"ParaValue": "",
"DueTime": "2018-11-17 00:06:00",
"BaseLineId": 332,
"NodeName": "trewtes",
"CycType": 0,
"DagType": 2,
"InGroupId": 1,
"Priority": 1,
"RerunAble": true,
"Status": 1,
"ResGroupIdentifier": "group_30004716",
"GwProcessId": "",
"PrgType": 10,
"Bizdate": "2018-11-16 00:00:00",
"ExecName": "/opt/taobao/tbdpapp/odpswrapper/odpswrapper.py",
"BeginWaitTimeTime": "",
"Gateway": "",
"GwLogFile": "",
"TaskType": 0
}
]
}
```

2.6.51. TestConnectivity

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to TestConnectivity.
BaseId	String	Yes	The ID of the Apsara Stack account.
Connection	String	Yes	The connection string of the data source.
Name	String	Yes	The name of the data source.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
SubType	String	Yes	The subtype of the data source. Currently, only MySQL is supported.
TenantId	Long	Yes	The ID of the tenant. Set the value to 1.
Type	String	Yes	The type of the data source.
Id	Long	No	The ID of the data source.
UseUserDefinedSecret	Boolean	No	Specifies whether to use the saved password for logging on to the data source.

Response parameters

Parameter	Type	Description
Data	Boolean	Indicates whether the data source is successfully connected.
ErrCode	Long	The error code.
ErrMsg	String	The error message.

Parameter	Type	Description
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=TestConnectivity
&BaseId=1486*****474
&Connection={"jdbcUrl":"jdbc:mysql://12.11.12.1:3306/data",
"username":"root ",
"password":"***",
"tag":"public"}
&Name="test"
&ProjectId=75059
&SubType=public
&TenantId=26788***178690
&Type=mysql
&<Common request parameters>
```

Sample success responses

```
{
"requestId": "0a98***7e2c95",
"data": null,
"errMsg": "Test for the data source connectivity failed:error message: Communications link failure\n\nThe last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server. error with code: PROJECT_DATASOURCE_CONN_ERROR",
"errCode": 610008
}
```

2.6.52. UpdateDQCFollower

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set this parameter to UpdateDQCFollower.
ProjectName	String	No	The name of the DataWorks workspace.

Parameter	Type	Required	Description
Id	Long	No	The ID of the subscription.
Follower	String	No	The object of receiving the subscribed messages. If you want to receive the messages by using emails or the SMS, specify your employee ID. If you want to receive the messages by using DingTalk, specify the webhook URL.
AlarmMode	Integer	No	The method of receiving alerts and notifications. A value of 1 indicates that you receive alerts and notifications by using emails. A value of 2 indicates that you receive alerts and notifications by using emails and the SMS. A value of 3 indicates that you receive alerts and notifications from the specified DingTalk group by using the webhook URL. A value of 4 indicates that you receive alerts and notifications from the specified DingTalk group and mention all members in the group.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	Boolean	Indicates whether the subscription information has been changed.

Sample request

```
http(s)://[Endpoint]/? Id=279580663
&ProjectName=autotest
&BlockType=1
&Checker=9
&Comment=Test
&CriticalThreshold=40
&EntityId=4003922
&ExpectValue=0
&MethodName=count/table_count
&Operator=50624
&Property=table_count
&PropertyType=table
&RuleType=0
&TemplateId=7
&Trend=abs
&WarningThreshold=20
&WhereCondition=60
&<Common request parameters>
```

Sample response

```
{
"ReturnValue": true,
"ReturnCode": "0"
}
```

2.6.53. UpdateDQCRule

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to UpdateDQCRule.
ProjectName	String	Yes	The name of the DataWorks workspace.
Id	Long	Yes	The ID of the Data Quality check (DQC) rule to change.

Parameter	Type	Required	Description
BlockType	Integer	No	The type of the DQC rule. Valid values: 0 and 1. The value 0 specifies a soft rule. The value 1 specifies a hard rule.
EntityId	Long	No	The ID of the partition filter expression.
Comment	String	No	The description of the DQC rule.
Checker	Integer	No	The ID of the checker.
ExpectValue	String	No	The expected value of the checker.
Trend	String	No	The trend of the expected value.
MethodName	String	No	The sampling method, such as max, table_count-count_distinct, and user_defined.
Operator	String	No	The user who is to change the DQC rule.
Property	String	No	The field of the DQC rule to change.
PropertyType	String	No	The type of the field.
RuleType	Integer	No	The type of the DQC rule. Valid values: 0 and 1. The value 0 specifies that the rule is defined by the system. The value 1 specifies that the rule is defined by users.
WhereCondition	String	No	The filtering condition. Write a SQL statement to specify the filtering condition.
CriticalThreshold	String	No	The error threshold.
WarningThreshold	String	No	The warning threshold.

Parameter	Type	Required	Description
TemplateId	Integer	No	The ID of the template to use for changing the DQC rule.

Response parameters

Parameter	Type	Description
ReturnCode	String	The return code.
ReturnValue	Boolean	Indicates whether the DQC rule has been changed.

Sample requests

```
http(s)://[Endpoint]/? Action=UpdateDQCFollower
&AlarmMode=1
&Follower=50624
&Id=1724
&ProjectName=autotest
&<Common request parameters>
```

Sample success responses

```
{
"ReturnValue": true,
"ReturnCode": "0"
}
```

2.6.54. UpdateResGroupGateWay

Request parameters

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. Set the value to UpdateResGroupGateWay.
BaseId	String	Yes	The ID of the Apsara Stack account.

Parameter	Type	Required	Description
Identifier	String	Yes	The identifier of the resource group.
Memory	Integer	Yes	The memory size of the gateway.
NodeName	String	Yes	The name of the node that runs the gateway.
ProjectId	Long	Yes	The ID of the DataWorks workspace.
TenantId	Long	Yes	The ID of the tenant.
VCpu	Integer	Yes	The number of cores in the vCPU of the gateway.

Response parameters

Parameter	Type	Description
Data	Boolean	Indicates whether the gateway configuration has been changed.
ErrCode	Long	The error code.
ErrMsg	String	The error message.
RequestId	String	The ID of the request.

Sample requests

```
http(s)://[Endpoint]/? Action=UpdateResGroupGateWay
&BaseId=1486821399873474
&Identifier=3907f9a58***8476b2506
&Memory=4
&NodeName=hostname
&ProjectId=75059
&TenantId=26788a***90
&VCpu=9
 &<Common request parameters>
```

Sample success responses

```
{
"requestId": "0bc174***5f15",
"data": true,
"errMsg": "success",
"errCode": 0
}
```

2.6.55. CheckMetaTable

Checks whether a table exists.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	No	The GUID of the table.

Response parameters

Parameter	Type	Description
Data	Boolean	Indicates whether the table exists.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("CheckMetaTable");
cr.putQueryParameter("TableGuid", tableGuid);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.56. GetMetaDB

Queries the information about a MaxCompute project.

Request parameters

Parameter	Type	Required	Description
DbGuid	STRING	No	The GUID of the MaxCompute project.

Response parameters

Parameter	Type	Description
AppGuid	STRING	The GUID of the MaxCompute project.
ProjectName	STRING	The name of the MaxCompute project.
ProjectNameCn	STRING	The description of the MaxCompute project.
OwnerId	STRING	The Apsara Stack tenant account of the MaxCompute project owner.
OwnerName	STRING	The name of the MaxCompute project owner.
CreateTime	STRING	The time when the MaxCompute project was created.
ModifyTime	STRING	The time when the MaxCompute project was modified.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("GetMetaDB");
cr.putQueryParameter("DbGuid", appGuid);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.57. GetMetaTable

Queries the detailed information about a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	No	The GUID of the table.

Response parameters

Parameter	Type	Description
AppGuid	STRING	The GUID of the MaxCompute project where the table resides.
SrcType	INTEGER	The source of the table. A value of 0 indicates MaxCompute.
LastDdlTime	STRING	The time when the table was last updated.
TableGuid	STRING	The GUID of the table.
OwnerId	STRING	The Apsara Stack tenant account of the table owner.
HeatDegree	INTEGER	The popularity of the table.

Parameter	Type	Description
Labels	STRING	The tags of the table. Multiple tags are separated with commas (,).
TableDesc	STRING	The description of the table.
HasPart	INTEGER	Indicates whether the table is a partitioned table. A value of 1 indicates that the table is a partitioned table.
IsVisible	INTEGER	Indicates whether the table is visible. A value of 1 indicates that the table is visible.
LifeCycle	INTEGER	The lifecycle of the table.
DataSize	LONG	The storage space occupied by the table.
TableName	STRING	The name of the table.
CreateTime	STRING	The time when the table was created.
LastModifyTime	STRING	The time when the table was last modified.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("GetMetaTable");
cr.putQueryParameter("TableGuid", tableGuid);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.58. GetMetaTableIntroWiki

Queries the instructions of the specified version on how to use a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.
WikiVersion	LONG	No	The version number of Wiki.

Response parameters

Parameter	Type	Description
GmtCreate	STRING	The time when the table was created.
GmtModified	STRING	The time when the table was modified.
CreatorName	STRING	The name of the table creator.
CreatorBaseId	STRING	The base ID of the table creator.
Content	STRING	The description of the table.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("GetMetaTableIntroWiki");
cr.putQueryParameter("TableGuid", tableGuid);
cr.putQueryParameter("WikiVersion", 2);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.59. ListMetaColumnLineage

Queries the lineage of a field.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.
ColumnName	STRING	Yes	The name of the field in the table.
Direction	STRING	Yes	Specifies whether to query the ancestor or descendant lineage of the field. A value of UP specifies that ancestor fields are to be queried, and a value of DOWN specifies that descendant fields are to be queried.

Response parameters

Parameter	Type	Description
TableName	STRING	The name of the ancestor or descendant table.
ColumnName	STRING	The name of the ancestor or descendant field.
ColumnGuid	STRING	The GUID of the ancestor or descendant field.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaColumnLineage");
cr.putQueryParameter("TableGuid", tableGuid);
cr.putQueryParameter("Direction", "DOWN");
cr.putQueryParameter("ColumnName", columnName);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.60. ListMetaTableChangeLog

Queries the change logs of a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.
StartDate	STRING	No	The beginning of the time range to query, in the yyyy-MM-dd format.
EndDate	STRING	No	The end of the time range to query, in the yyyy-MM-dd format.
ChangeType	STRING	No	The change type. Valid values: *CREATE_TABLE*, *ALTER_TABLE*, *DROP_TABLE*, *ADD_PARTITION*, and *DROP_PARTITION*.
ObjectType	STRING	No	The entity on which the change was made. Valid values: *TABLE* and *PARTITION*.

Parameter	Type	Required	Description
PageNum	INTEGER	No	The number of the page to return.
PageSize	INTEGER	No	The number of entries to return on each page.

Response parameters

Parameter	Type	Description
GmtCreate	STRING	The time when the table was created.
GmtModified	STRING	The time when the table was modified.
Guid	STRING	The name of the table creator.
ChangeType	STRING	The change type.
ObjectType	STRING	The entity on which the change was made.
Operator	STRING	The name of the operator.
ChangeContent	STRING	The change description.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaTableChangeLog");
cr.putQueryParameter("TableGuid", tableGuid);
cr.putQueryParameter("StartDate", "2020-03-22");
cr.putQueryParameter("EndDate", "2020-03-26");
cr.putQueryParameter("ChangeType", "ADD_PARTITION");
cr.putQueryParameter("ObjectType", "PARTITION");
cr.putQueryParameter("PageNum", String.valueOf(1));
cr.putQueryParameter("PageSize", String.valueOf(3));
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.61. ListMetaTableColumn

Queries the information about fields in a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	String	Yes	The GUID of the table.

Response parameters

Parameter	Type	Description
TableGuid	STRING	The GUID of the table.
TableName	STRING	The name of the table.
ColumnGuid	STRING	The GUID of the field.
ColumnName	STRING	The name of the field.
ColumnType	STRING	The type of the field.

Parameter	Type	Description
SeqNumber	INTEGER	The sequence number of the field.
IsPartitionCol	INTEGER	Indicates whether the field is a partition field. A value of 1 indicates that the field is a partition field.
IsPrimaryKey	INTEGER	Indicates whether the field is a primary key. A value of 1 indicates that the field is a primary key.
IsNullable	INTEGER	Indicates whether the field can be null.
Comment	STRING	The description of the field.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaTableColumn");
cr.putQueryParameter("TableGuid", tableGuid);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.62. ListMetaTableImpact

Queries the impact analysis result of a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.

Parameter	Type	Required	Description
ColumnName	STRING	No	The name of the field.
Direction	STRING	Yes	Specifies whether to query the impacts on ancestor or descendant tables. Valid values: *OUT* and *PUT*.

Response parameters

Parameter	Type	Description
TableGuid	STRING	The GUID of the table.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaTableColumn");
cr.putQueryParameter("TableGuid", tableGuid);
cr.putQueryParameter("Direction", "OUTPUT");
cr.putQueryParameter("ColumnName", columnName);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.63. ListMetaTableIntroWiki

Queries the instructions on how to use a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.

Response parameters

Parameter	Type	Description
TableGuid	STRING	The GUID of the table.
GmtCreate	STRING	The time when the table was created.
GmtModified	STRING	The time when the table was modified.
Creator	STRING	The base ID of the table creator.
CreatorName	STRING	The name of the table creator.
CreatorBaseId	STRING	The base ID of the table creator.
Version	LONG	The version number of Wiki.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaTableIntroWiki");
cr.putQueryParameter("TableGuid", tableGuid);
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.64. ListMetaTableLineage

Queries the lineage of a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.

Parameter	Type	Required	Description
Direction	STRING	Yes	Specifies whether to query the ancestor or descendant lineage of the table. Valid values: *UP*, *DOWN*, and *ALL*.
PageNum	INTEGER	No	The number of the page to return.
PageSize	INTEGER	No	The number of entries to return on each page.

Response parameters

Parameter	Type	Description
DstTableGuid	STRING	The GUID of the destination table.
SrcTableGuid	STRING	The GUID of the source table.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaTableLineage");
cr.putQueryParameter("TableGuid", tableGuid);
cr.putQueryParameter("Direction", "ALL");
cr.putQueryParameter("PageNum", String.valueOf(1));
cr.putQueryParameter("PageSize", String.valueOf(3));
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.65. ListMetaTableOutput

Queries the information about the node instances that generate data for a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.
StartDate	STRING	No	The beginning of the time range to query, in the yyyy-MM-dd format.
EndDate	STRING	No	The end of the time range to query, in the yyyy-MM-dd format.
PageNum	INTEGER	No	The number of the page to return.
PageSize	INTEGER	No	The number of entries to return on each page.

Response parameters

Parameter	Type	Description
TableGuid	STRING	The GUID of the table.
ProjectId	LONG	The ID of the DataWorks workspace to which the node belongs.
TaskId	STRING	The ID of the node.
TaskInstId	LONG	The ID of the node instance.
StartTime	STRING	The time when the node instance was started.
EndTime	STRING	The time when the node instance was ended.
ExecuteTime	STRING	The time taken to run the node instance.
WaitTime	STRING	The period during which the node instance was waiting to be run.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaTableOutput");
cr.putQueryParameter("TableGuid", tableGuid);
cr.putQueryParameter("StartDate", "2020-03-22");
cr.putQueryParameter("EndDate", "2020-03-26");
cr.putQueryParameter("PageNum", String.valueOf(2));
cr.putQueryParameter("PageSize", String.valueOf(2));
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.66. ListMetaTablePartition

Queries the partition information of a table.

Request parameters

Parameter	Type	Required	Description
TableGuid	STRING	Yes	The GUID of the table.
PageNum	INTEGER	No	The number of the page to return.
PageSize	INTEGER	No	The number of entries to return on each page.

Response parameters

Parameter	Type	Description
AppGuid	STRING	The GUID of the MaxCompute project.
TableGuid	STRING	The GUID of the table.
TableName	STRING	The name of the table.

Parameter	Type	Description
PartitionGuid	STRING	The GUID of the partition.
PartitionName	STRING	The name of the partition.
CreateTime	STRING	The time when the partition was created.
ModifyTime	STRING	The time when the partition was modified.
DataSize	LONG	The storage space occupied by the partition.
Records	LONG	The number of records in the partition.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("ListMetaTablePartition");
cr.putQueryParameter("TableGuid", tableGuid);
cr.putQueryParameter("PageNum", String.valueOf(2));
cr.putQueryParameter("PageSize", String.valueOf(2));
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.67. SearchMetaTables

Queries MaxCompute tables.

Request parameters

Parameter	Type	Required	Description

Parameter	Type	Required	Description
Keyword	STRING	Yes	The keyword used to query MaxCompute tables.
PageNum	INTEGER	No	The number of the page to return.
PageSize	INTEGER	No	The number of entries to return on each page.

Response parameters

Parameter	Type	Description
AppGuid	STRING	The GUID of the MaxCompute project.
TableGuid	STRING	The GUID of the table.
TableName	STRING	The name of the table.
CreateTime	STRING	The time when the table was created.
LastDdlTime	STRING	The time when the table schema was last changed.
LastModifyTime	STRING	The time when the table was last updated.
OwnerId	STRING	The base ID of the table owner.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("SearchMetaTables");
cr.putQueryParameter("Keyword", tableGuid);
cr.putQueryParameter("PageNum", String.valueOf(1));
cr.putQueryParameter("PageSize", String.valueOf(2));
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.68. GetStorageAndCalcMetrics

Queries the storage and computing data of a specific project or the current Apsara Stack tenant account.

Request parameters

Parameter	Type	Required	Description
AppGuids	STRING	No	The GUIDs of the MaxCompute projects. The GUIDs are separated with commas (,).
DateString	STRING	No	The query date. The default value is the data timestamp.
PageSize	INTEGER	No	The number of entries to return on each page.

Response parameters

Parameter	Type	Description
TotalStorage	LONG	The name of the table.
TotalCpuCost	LONG	The GUID of the table.

Sample code

```
private String product = "dataworks-private-cloud";
private String version = "2019-01-17";
private String appGuid = "odps.dw_meta";
private String tableGuid = "odps.dw_meta.dwd_d_odps_table";
private String columnName = "table_comment";
CommonRequest cr = new CommonRequest();
cr.setProduct(product);
cr.setVersion(version);
cr.setMethod(MethodType.GET);
cr.setAction("GetStorageAndCalcMetrics");
cr.putQueryParameter("AppGuids", appGuid);
cr.putQueryParameter("DateString", '2020-01-01');
CommonResponse cresp = client.getCommonResponse(cr);
assertNotNull(cresp);
assertNotNull(cresp.getData());
```

2.6.69. GetInstanceStatistic

Queries instance statistics.

Request parameters

Parameter	Required	Type	Description
Bizdate	No	STRING	The data timestamp, in the yyyy-MM-dd format.
AppId	No	LONG	The ID of the workspace.

Sample response

```
{
 "ReturnErrorSolution": "",
 "ReturnCode": "0",
 "RequestId": "83190FF7-8EAF-44DB-9170-AC0AXXXX",
 "ReturnMessage": "",
 "ReturnValue": {
  "TotalCount": 15,
  "RunningCount": 0,
  "FailureCount": 0,
  "NotRunCount": 15,
  "SuccessCount": 0,
  "WaitTimeCount": 0,
  "WaitResourceCount": 0
 },
 "Success": true
}
```


Sample code

```
@Test
public void testGetInstanceStatistic() throws ClientException {
  CommonRpcRequest commonRpcRequest = new CommonRpcRequest(popProduct, popVersion, "GetInsta
nceStatistic");
  commonRpcRequest.setMethod(MethodType.POST);
  String projectId = getTestProperties("projectId");
  String bizdate = getTestProperties("bizdate");
  commonRpcRequest.putQueryParameter("AppId", projectId);
  commonRpcRequest.putQueryParameter("Bizdate", bizdate);
  HttpResponse response = client.doAction(commonRpcRequest);
  System.out.println(GsonUtils.gson.toJson(response.getHttpContentString()));
  }
```

2.6.70. GetInstanceLog

Queries the operational logs of a node instance.

Request parameters

Parameter	Required	Type	Description
InstanceId	No	LONG	The ID of the instance.

Parameter	Required	Type	Description
ProjectEnv	Yes	STRING	The environment of the node instance. Valid values: PROD and DEV.

Sample response

```
{
 "ReturnErrorSolution": "",
 "ReturnCode": "0",
 "RequestId": "CEC994D9-45BD-4531-B2CE-6XXXX",
 "ReturnMessage": "",
 "ReturnValue": "No operational logs are generated.",
 "Success": true
}
```

Sample code

```
@Test
  public void testGetInstanceLog() throws ClientException {
    CommonRpcRequest commonRpcRequest = new CommonRpcRequest(popProduct, popVersion, "GetIns
tanceLog");
    commonRpcRequest.setMethod(MethodType.POST);
    Long instanceId = Long.valueOf(getTestProperties("instanceId"));
    String projectEnv = getTestProperties("projectEnv");
    commonRpcRequest.putQueryParameter("InstanceId", instanceId);
    commonRpcRequest.putQueryParameter("ProjectEnv", projectEnv);
    HttpResponse response = client.doAction(commonRpcRequest);
    System.out.println(GsonUtils.gson.toJson(response.getHttpContentString()));
  }
```

2.6.71. GetNodeCode

Queries the code of a node.

Request parameters

Parameter	Required	Type	Description
NodeId	Yes	LONG	The ID of the node.

Parameter	Required	Type	Description
ProjectEnv	Yes	STRING	The environment of the node. Valid values: PROD and DEV. Default value: PROD.

Sample response

```
{
 "ReturnErrorSolution": "",
 "ReturnCode": "0",
 "RequestId": "30DB3056-F6F1-4A05-88EE-5XXXX",
 "ReturnMessage": "",
 "ReturnValue": "echo 'hello'"
}
```

Sample code

```
  @Test
  public void testGetNodeCode() throws ClientException {
    CommonRpcRequest commonRpcRequest = new CommonRpcRequest(popProduct, popVersion, "GetNo
deCode");
    commonRpcRequest.setMethod(MethodType.POST);
    Long nodeId = Long.valueOf(getTestProperties("nodeId"));
    String projectEnv = getTestProperties("projectEnv");
    commonRpcRequest.putQueryParameter("NodeId", nodeId);
    commonRpcRequest.putQueryParameter("ProjectEnv", projectEnv);
    HttpResponse response = client.doAction(commonRpcRequest);
    System.out.println(GsonUtils.gson.toJson(response.getHttpContentString()));
  }
```

2.6.72. QueryInstances

Queries instances.

Request parameters

Parameter	Required	Type	Description
Bizdate	No	STRING	The data timestamp, in the yyyy-MM-dd format.

Parameter	Required	Type	Description
ProjectId	No	LONG	The ID of the workspace.
NodeId	Yes	LONG	The ID of the node.
NodeName	Yes	STRING	The name of the node.
TaskStatus	Yes	STRING	The status of the node. Valid values: NOT_RUN, WAIT_TIME, WAIT_RESOURCE, RUNNING, CHECKING, CHECKING_CONDITION, FAILURE, and SUCCESS.
ProjectEnv	Yes	STRING	The environment of the node. Valid values: PROD and DEV.

Sample response

```
{
 "ReturnErrorSolution": "",
 "ReturnCode": "0",
 "RequestId": "7AFA7207-2F44-4ECC-AE77-XXXX",
 "ReturnMessage": "",
 "ReturnValue": [
  {
   "Status": "SUCCESS",
   "ModifyTime": 15881766XXXX,
   "InstanceId": 70652XXXX,
   "DagType": "DAILY",
   "NodeName": "Delay node",
   "CreateTime": 1588088309461,
   "CycTime": 1588176600000,
   "BeginWaitTimeTime": 1588176XXXX,
   "DagId": 700126518687,
   "Bizdate": 1588089600000,
   "BeginRunningTime": 1588176671359,
   "FinishTime": 1588176680949,
   "NodeId": 700001891566,
   "BeginWaitResTime": 1588176601707
  },
  {
   "Status": "SUCCESS",
   "ModifyTime": 1588090445868,
   "InstanceId": 706518829092,
   "DagType": "DAILY",
   "NodeName": "Delay node",
   "CreateTime": 1588001922168,
   "CycTime": 1588090200000,
   "BeginWaitTimeTime": 1588089826198,
   "DagId": 700126318686,
   "Bizdate": 1588003200000,
   "BeginRunningTime": 1588090305062,
   "FinishTime": 1588090445868,
   "NodeId": 700001891566,
   "BeginWaitResTime": 1588090203188
  }
]
```

Sample code

```
@Test
public void testQueryInstances() throws ClientException {
    CommonRpcRequest commonRpcRequest = new CommonRpcRequest(popProduct, popVersion, "QueryInstances");
    commonRpcRequest.setMethod(MethodType.POST);
    String projectId = getTestProperties("projectId");
    String projectEnv = getTestProperties("projectEnv");
    String bizdate = getTestProperties("bizdate");
    String pageNumber = getTestProperties("pageNumber");
    String nodeId = getTestProperties("nodeId");
    String nodeName = getTestProperties("nodeName");
    String taskStatus = getTestProperties("taskStatus");
    commonRpcRequest.putQueryParameter("Bizdate", bizdate);
    commonRpcRequest.putQueryParameter("ProjectId", projectId);
    commonRpcRequest.putQueryParameter("NodeId", nodeId);
    commonRpcRequest.putQueryParameter("PageNumber", pageNumber);
    commonRpcRequest.putQueryParameter("NodeName", nodeName);
    commonRpcRequest.putQueryParameter("ProjectEnv", projectEnv);
    commonRpcRequest.putQueryParameter("TaskStatus", taskStatus);
    HttpResponse response = client.doAction(commonRpcRequest);
    System.out.println(GsonUtils.gson.toJson(response.getHttpContentString()));
}
```

2.7. Obtain an AccessKey pair

You can obtain an AccessKey pair from the Apsara Stack Cloud Management (ASCM) console.

Obtain the AccessKey pair of an organization

To obtain the AccessKey pair of an organization, follow these steps:

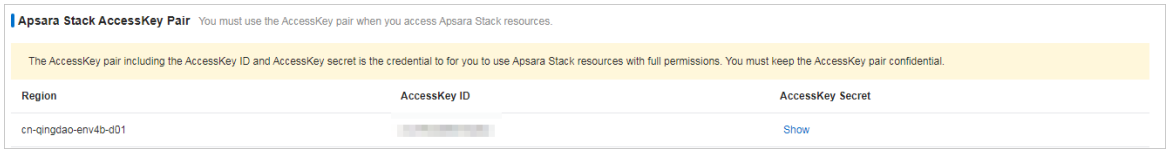
1. Log on to the ASCM console as an administrator.
2. In the top navigation bar, click **Enterprise**.
3. In the left-side navigation pane of the **Enterprise** page, click **Organizations**.
4. In the Organizations navigation tree, click the icon on the right of an organization.
5. Choose **AccessKey** from the shortcut menu.
6. In the AccessKey dialog box that appears, view the AccessKey pair of the organization.

> **Note** An AccessKey pair is automatically allocated to a level-1 organization. Subordinate organizations use the same AccessKey pair as their level-1 organization.

Obtain the AccessKey pair of a personal account

To obtain the AccessKey pair of a personal account, follow these steps:

1. Log on to the ASCM console as an administrator.
2. In the upper-right corner of the homepage, move the pointer over the user profile picture and click **User Information**.
3. In the **Apsara Stack AccessKey Pair** section of the User Information page, view your AccessKey pair.



> **Note** The AccessKey pair consists of an AccessKey ID and an AccessKey secret. They allow you to access Apsara Stack resources with full permissions for your account. You must keep them confidential.

3.Realtime Compute

3.1. Flink SQL

3.1.1. Overview

As a programming language that conforms to standard SQL semantics, Flink SQL is designed to simplify the computational model, making it easy for users to use Realtime Compute. Flink SQL has the following advantages:

- Low development cost. Flink SQL enables you to implement complex and varied business logic by using simple SQL statements instead of thousands of lines of Apache Storm code.
- Low maintenance cost. Flink SQL eliminates the need for you to maintain thousands of lines of Storm code.

When you use Apache Storm code, system exceptions, errors, or even breakdown may occur and cause data inaccuracies.

Flink SQL features specified semantics to resolve the preceding issues. Realtime Compute has the following advantages:

- Data accuracy. Realtime Compute provides unique exactly-once semantics to ensure data accuracy in various failover scenarios.
- Performance optimization. Realtime Compute integrates SQL optimization logic to translate SQL code into underlying Apache Storm code that can be used by universal developers. This helps to improve SQL code compilation efficiency.
- Programming extensibility. Realtime Compute supports user-defined functions (UDFs). You can customize functions to implement a custom business logic or a logic that cannot be achieved by SQL statements.
- System scalability. Realtime Compute supports auto scaling upon data surges (for example, during the Double 11) while ensuring job and system stability.

3.1.2. Keywords

This topic describes the keywords in Realtime Compute.

Realtime Compute keywords

Common keywords

Keyword type	Keywords
Data type	VARCHAR, INT, BIGINT, DOUBLE, DATE, BOOLEAN, TINYINT, SMALLINT, FLOAT, DECIMAL, and VARBINARY
DDL statement	CREATE TABLE, CREATE FUNCTION, and CREATE VIEW
DML statement	INSERT INTO
SELECT clause	SELECT FROM, WHERE, GROUP BY, and JOIN

Naming conventions

Names of source tables, result tables, views, and aliases must follow the standard database naming conventions. The names must start with a letter, and can only contain letters, numbers, and underscores (_).

Reserved keywords

The following strings are reserved as keywords for later use. If you want to use any of the following keywords as a field name, enclose the keyword in backticks (`), for example, `` `value` ``.

A, ABS, ABSOLUTE, ACTION, ADA, ADD, ADMIN, AFTER, ALL, ALLOCATE, ALLOW, ALTER, ALWAYS, AND, ANY, ARE, ARRAY, AS, ASC, ASENSITIVE, ASSERTION, ASSIGNMENT, ASYMMETRIC, AT, ATOMIC, ATTRIBUTE, ATTRIBUTES, AUTHORIZATION, AVG, BEFORE, BEGIN, BERNOULLI, BETWEEN, BIGINT, BINARY, BIT, BLOB, BOOLEAN, BOTH, BREADTH, BY, C, CALL, CALLED, CARDINALITY, CASCADE, CASCADED, CASE, CAST, CATALOG, CATALOG_NAME, CEIL, CEILING, CENTURY, CHAIN, CHAR, CHARACTER, CHARACTERISTICTS, CHARACTERS, CHARACTER_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_NAME, CHARACTER_SET_SCHEMA, CHAR_LENGTH, CHECK, CLASS_ORIGIN, CLOB, CLOSE, COALESCE, COBOL, COLLATE, COLLATION, COLLATION_CATALOG, COLLATION_NAME, COLLATION_SCHEMA, COLLECT, COLUMN, COLUMN_NAME, COMMAND_FUNCTION, COMMAND_FUNCTION_CODE, COMMIT, COMMITTED, CONDITION, CONDITION_NUMBER, CONNECT, CONNECTION, CONNECTION_NAME, CONSTRAINT, CONSTRAINTS, CONSTRAINT_CATALOG, CONSTRAINT_NAME, CONSTRAINT_SCHEMA, CONSTRUCTOR, CONTAINS, CONTINUE, CONVERT, CORR, CORRESPONDING, COUNT, COVAR_POP, COVAR_SAMP, CREATE, CROSS, CUBE, CUME_DIST, CURRENT, CURRENT_CATALOG, CURRENT_DATE, CURRENT_DEFAULT_TRANSFORM_GROUP, CURRENT_PATH, CURRENT_ROLE, CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_TRANSFORM_GROUP_FOR_TYPE, CURRENT_USER, CURSOR, CURSOR_NAME, CYCLE, DATA, DATABASE, DATE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, DAY, DEALLOCATE, DEC, DECADE, DECIMAL, DECLARE, DEFAULT, DEFAULTS, DEFERRABLE, DEFERRED, DEFINED, DEFINER, DEGREE, DELETE, DENSE_RANK, DEPTH, DEREF, DERIVED, DESC, DESCRIBE, DESCRIPTION, DESCRIPTOR, DETERMINISTIC, DIAGNOSTICS, DISALLOW, DISCONNECT, DISPATCH, DISTINCT, DOMAIN, DOUBLE, DOW, DOY, DROP, DYNAMIC, DYNAMIC_FUNCTION, DYNAMIC_FUNCTION_CODE, EACH, ELEMENT, ELSE, END, END-EXEC, EPOCH, EQUALS, ESCAPE, EVERY, EXCEPT, EXCEPTION, EXCLUDE, EXCLUDING, EXEC, EXECUTE, EXISTS, EXP, EXPLAIN, EXTEND, EXTERNAL, EXTRACT, FALSE, FETCH, FILTER, FINAL, FIRST, FIRST_VALUE, FLOAT, FLOOR, FOLLOWING, FOR, FOREIGN, FORTRAN, FOUND, FRAC_SECOND, FREE, FROM, FULL, FUNCTION, FUSION, G, GENERAL, GENERATED, GET, GLOBAL, GO, GOTO, GRANT, GRANTED, GROUP, GROUPING, HAVING, HIERARCHY, HOLD, HOUR, IDENTITY, IMMEDIATE, IMPLEMENTATION, IMPORT, IN, INCLUDING, INCREMENT, INDICATOR, INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INSTANCE, INSTANTIABLE, INT, INTEGER, INTERSECT, INTERSECTION, INTERVAL, INTO, INVOKER, IS, ISOLATION, JAVA, JOIN, K, KEY, KEY_MEMBER, KEY_TYPE, LABEL, LANGUAGE, LARGE, LAST, LAST_VALUE, LATERAL, LEADING, LEFT, LENGTH, LEVEL, LIBRARY, LIKE, LIMIT, LN, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR, LOWER, M, MAP, MATCH, MATCHED, MAX, MAXVALUE, MEMBER, MERGE, MESSAGE_LENGTH, MESSAGE_OCTET_LENGTH, MESSAGE_TEXT, METHOD, MICROSECOND, MILLENNIUM, MIN, MINUTE, MINVALUE, MOD, MODIFIES, MODULE, MONTH, MORE, MULTISET, MUMPS, NAME, NAMES, NATIONAL, NATURAL, NCHAR, NCLOB, NESTING, NEW, NEXT, NO, NONE, NORMALIZE, NORMALIZED, NOT, NULL, NULLABLE, NULLIF, NULLS, NUMBER, NUMERIC, OBJECT, OCTETS,

OCTET_LENGTH, OF, OFFSET, OLD, ON, ONLY, OPEN, OPTION, OPTIONS, OR, ORDER, ORDERING, ORDINALITY, OTHERS, OUT, OUTER, OUTPUT, OVER, OVERLAPS, OVERLAY, OVERRIDING, PAD, PARAMETER, PARAMETER_MODE, PARAMETER_NAME, PARAMETER_ORDINAL_POSITION, PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_NAME, PARAMETER_SPECIFIC_SCHEMA, PARTIAL, PARTITION, PASCAL, PASSTHROUGH, PATH, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK, PLACING, PLAN, PLI, POSITION, POWER, PRECEDING, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR, PRIVILEGES, PROCEDURE, PUBLIC, QUARTER, RANGE, RANK, READ, READS, REAL, RECURSIVE, REF, REFERENCES, REFERENCING, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY, RELATIVE, RELEASE, REPEATABLE, RESET, RESTART, RESTRICT, RESULT, RETURN, RETURNED_CARDINALITY, RETURNED_LENGTH, RETURNED_OCTET_LENGTH, RETURNED_SQLSTATE, RETURNS, REVOKE, RIGHT, ROLE, ROLLBACK, ROLLUP, ROUTINE, ROUTINE_CATALOG, ROUTINE_NAME, ROUTINE_SCHEMA, ROW, ROWS, ROW_COUNT, ROW_NUMBER, SAVEPOINT, SCALE, SCHEMA, SCHEMA_NAME, SCOPE, SCOPE_CATALOGS, SCOPE_NAME, SCOPE_SCHEMA, SCROLL, SEARCH, SECOND, SECTION, SECURITY, SELECT, SELF, SENSITIVE, SEQUENCE, SERIALIZABLE, SERVER, SERVER_NAME, SESSION, SESSION_USER, SET, SETS, SIMILAR, SIMPLE, SIZE, SMALLINT, SOME, SOURCE, SPACE, SPECIFIC, SPECIFICTYPE, SPECIFIC_NAME, SQL, SQLEXCEPTION, SQLSTATE, SQLWARNING, SQL_TSI_DAY, SQL_TSI_FRAC_SECOND, SQL_TSI_HOUR, SQL_TSI_MICROSECOND, SQL_TSI_MINUTE, SQL_TSI_MONTH, SQL_TSI_QUARTER, SQL_TSI_SECOND, SQL_TSI_WEEK, SQL_TSI_YEAR, SQRT, START, STATE, STATEMENT, STATIC, STDDEV_POP, STDDEV_SAMP, STREAM, STRUCTURE, STYLE, SUBCLASS_ORIGIN, SUBMULTISET, SUBSTITUTE, SUBSTRING, SUM, SYMMETRIC, SYSTEM, SYSTEM_USER, TABLE, TABLESAMPLE, TABLE_NAME, TEMPORARY, THEN, TIES, TIME, TIMESTAMP, TIMESTAMPADD, TIMESTAMPDIFF, TIMEZONE_HOUR, TIMEZONE_MINUTE, TINYINT, TO, TOP_LEVEL_COUNT, TRAILING, TRANSACTION, TRANSACTIONS_ACTIVE, TRANSACTIONS_COMMITTED, TRANSACTIONS_ROLLED_BACK, TRANSFORM, TRANSFORMS, TRANSLATE, TRANSLATION, TREAT, TRIGGER, TRIGGER_CATALOG, TRIGGER_NAME, TRIGGER_SCHEMA, TRIM, TRUE, TYPE, UESCAPE, UNBOUNDED, UNCOMMITTED, UNDER, UNION, UNIQUE, UNKNOWN, UNNAMED, UNNEST, UPDATE, UPPER, UPSERT, USAGE, USER, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_CODE, USER_DEFINED_TYPE_NAME, USER_DEFINED_TYPE_SCHEMA, USING, VALUE, VALUES, VARBINARY, VARCHAR, VARYING, VAR_POP, VAR_SAMP, VERSION, VIEW, WEEK, WHEN, WHENEVER, WHERE, WIDTH_BUCKET, WINDOW, WITH, WITHIN, WITHOUT, WORK, WRAPPER, WRITE, XML, YEAR, ZONE

3.1.3. Terms

3.1.3.1. Watermark

Realtime Compute aggregates data by using window functions based on a time attribute. To use window functions based on event time, you must define a watermark in the DDL statement of a source table.

Watermarking is a mechanism in Flink to measure progress in event time. A watermark provides a time attribute by carrying a timestamp. Flink provides the following statement to define a watermark:

> **Note** For more information about time attributes supported by Realtime Compute, see Time attributes.

```
WATERMARK [watermarkName] FOR <rowtime_field> AS withOffset(<rowtime_field>, offset)
```

Parameter	Description
watermarkName	Specifies the name of a watermark. This parameter is optional.
<rowtime_field>	Specifies a column that is used to generate a watermark. The data type of the column must be TIMESTAMP. The column is labeled as the event time column. You can use this event time column to define windows in queries.
withOffset	Specifies the strategy of generating a watermark. The value for a watermark is obtained based on `<rowtime_field> - offset`. In `withOffset`, you must define `<rowtime_field>` as the first parameter.
offset	Specifies the offset between a watermark value and an event time, which is measured in milliseconds.

The time when a data record is generated is indicated by a field. For example, a table has a `rowtime` column of the TIMESTAMP data type, and one value is `1501750584000 (2017-08-03 08:56:24.000)`. Based on the `rowtime` column, you can define a watermark with a four-second offset as follows:

```
WATERMARK FOR rowtime AS withOffset(rowtime, 4000)
```

The watermark value of the specified data record is obtained: `1501750584000 - 4000 = 1501750580000 (2017-08-03 08:56:20.000)`. The watermark value indicates that all data with the timestamps earlier than `1501750580000 (2017-08-03 08:56:20.000)` has arrived.

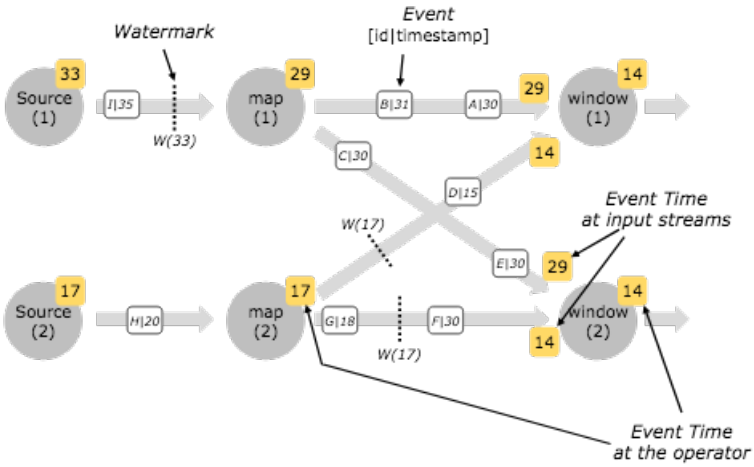
> **Note**
>
> - When you define a watermark based on event time, the data type of the `rowtime` column must be TIMESTAMP. Currently, Realtime Compute only supports 13-bit Unix timestamps measured in milliseconds. If the data type of a rowtime column is not TIMESTAMP or a Unix timestamp is not 13 digits in length, we recommend that you use a Computed column to convert data types.
> - Event time and processing time can only be declared in the DDL statement of a source table.

Summary

- A watermark indicates that all the events with the timestamps t' earlier than the watermark `t` have occurred. After a watermark `t` arrives at an operator, all subsequent records with timestamps earlier

than the watermark `t` are discarded. This is the current watermarking mechanism in Realtime Compute. However, in the future, improvements to Realtime Compute will allow you to update subsequent data.

- The watermarking mechanism is crucial for handling out-of-order data streams. With watermarks, real-time computing results are accurate even if data streams arrive out of order.
- When an operator has multiple input data streams, the event time of the operator equals the minimum event time of the data streams.



3.1.3.2. Computed column

You can calculate a value for a computed column by using data from other columns. If a source table does not have a column of the TIMESTAMP data type, you can use a computed column expression to convert data types.

Definition

A computed column is a virtual column that is not physically stored in a table. You can create computed columns by using expressions, built-in functions, or user-defined functions. In Flink SQL, a computed column can be used as other columns that are physically stored in a table.

Usage

Currently, the data type of the event time (also known as rowtime) column in a Watermark must be TIMESTAMP. We are working on support for the LONG data type. You can only define a watermark in the DDL statement of a source table. If a source table does not have a column of the TIMESTAMP data type, you can use a computed column expression to convert data types.

Syntax

```
column_name AS computed_column_expression
```

Example

The data type of the rowtime column in a watermark must be TIMESTAMP. Currently, Realtime Compute only supports 13-bit Unix timestamps measured in milliseconds. If the TIME column in a DataHub source table is defined as 16-bit Unix timestamps measured in microseconds, you can use a computed column expression to convert data types. An example is described as follows:

```
CREATE TABLE test_stream(
 a INT,
 b BIGINT,
 `TIME` BIGINT,
 Ts AS TO_TIMESTAMP (TIME/1000), -- Converts 16-bit timestamps to 13-bit timestamps by using a compute
d column expression.
 WATERMARK FOR ts AS withOffset(ts, 10000)
) WITH (
 type = 'datahub',
 ...
);
```

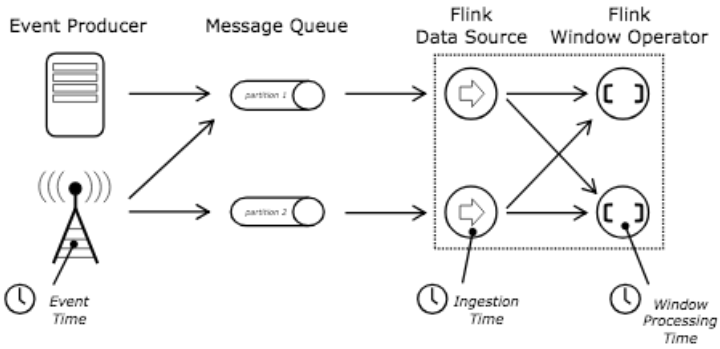
In this example, the data type of the `TIME` column in the source table is BIGINT. A computed column expression is created to convert the `TIME` column of the BIGINT data type to the `ts` column of the TIMESTAMP data type. The `ts` column is used as the rowtime of a watermark.

3.1.3.3. Time attributes

This topic describes two time attributes that are supported by Flink SQL: event time and processing time.

Event time and processing time are described as follows:

- Event time: specifies the time at which an event is produced. The event time column must be included in the schema.
- Processing time: specifies the time at which the system processes an event.



Ingestion time and processing time are the time attributes automatically generated by the system for a streaming data record and are not changeable. Event time is the time attribute carried by a streaming data record itself. Assume that data whose event time is t1 is written to Partition 1 and data whose event time is t2 is written to Partition 2. The event time t1 is earlier than t2. However, due to data disorder or network jitter, Realtime Compute may first read and process the data with an event time of t2.

Event time

Event time is also known as rowtime. The event time attribute must be declared in the DDL statement for creating a source table. You can declare a field in the source table as event time. Currently, only the TIMESTAMP data type is supported for a rowtime field. We are working on providing support for the LONG data type. If the data type of the specified field is not TIMESTAMP, you must create a computed column of the TIMESTAMP data type from existing fields. For more information about computed columns, see Computed column.

Due to data disorder or network jitter, the sequence in which Realtime Compute reads and processes data may differ from the sequence in which data arrives. Therefore, after you define the rowtime field, you must define a Watermark.

You can perform event time-based window function operations as follows:

```
CREATE TABLE tt_stream(
 a VARCHAR,
 b VARCHAR,
 c TIMESTAMP,
 WATERMARK wk1 FOR c as withOffset(c, 1000)  -- Defines a watermark.
) WITH (
 type = 'sls',
 topic = 'yourTopicName',
 accessId = 'yourAccessId',
 accessKey = 'yourAccessSecret'
);
CREATE TABLE rds_output(
 id VARCHAR,
 c TIMESTAMP,
 f TIMESTAMP,
 cnt BIGINT
) WITH (
 type = 'rds',
 url = 'jdbc:mysql://****3306/test',
 tableName='yourTableName',
 userName = 'yourUserName',
 password = 'yourPassword'
);
INSERT INTO rds_output
SELECT a AS id,
   SESSION_START(c, INTERVAL '1' SECOND) AS c,
   CAST(SESSION_END(c, INTERVAL '1' SECOND) AS TIMESTAMP) AS f,
   COUNT(a) AS cnt
FROM tt_stream
GROUP BY SESSION(c, INTERVAL '1' SECOND), a
```

Processing time

Processing time is generated by the system and is not included in your original data. Therefore, you need to define a processing time field in a DDL statement.

```
filedName as PROCTIME()
```

You can aggregate data by using a window function based on processing time. An example is described as follows:

```
CREATE TABLE mq_stream (
  a VARCHAR,
  b VARCHAR,
  c BIGINT,
  d AS PROCTIME()   --- Defines a processing time field.
) WITH (
  type = 'rds',
  topic = 'yourTopic',
  accessId = 'yourAccessId',
  accessKey = 'yourAccessSecret'
);
CREATE TABLE rds_output (
  id VARCHAR,
  c TIMESTAMP,
  f TIMESTAMP,
  cnt BIGINT
) with (
  type = 'rds',
  url = 'yourDatebaseURL',
  tableName='yourDatabaseTableName',
  userName = 'yourUserName',
  password = 'yourPassword'
);
INSERT INTO rds_output
SELECT a AS id,
    SESSION_START(d, INTERVAL '1' SECOND) AS c,
    SESSION_END(d, INTERVAL '1' SECOND) AS f,
    COUNT(a) AS cnt
FROM mq_stream
GROUP BY SESSION(d, INTERVAL '1' SECOND), a
```

Expiration of time attributes

A time attribute field no longer indicates a time attribute if you perform the following two operations:

- GROUP BY operations (not for window functions)
- JOIN operations of two data streams

Errors will occur if you perform time-based window function operations after the preceding operations. For example, the error message `org. apache. flink. table. api. ValidationException: Window can only be defined over a time attribute column.` may appear.

3.1.4. Data types

3.1.4.1. Data types

This topic describes data types supported by Realtime Compute.

Supported data types

Data type	Description	Value range
VARCHAR	Character string with a changeable length	Maximum storage capacity: 4 MB
BOOLEAN	Logical value	Valid values: TRUE, FALSE, and UNKNOWN
TINYINT	Tiny integer (one byte)	-128 to 127
SMALLINT	Small integer (two bytes)	-32768 to 32767
INT	Integer (four bytes)	-2147483648 to 2147483647
BIGINT	Big integer (eight bytes)	-9223372036854775808 to 9223372036854775807
FLOAT	Four-byte floating point number	Six digits of precision
DECIMAL	Decimal	An example value for `DECIMAL(5,2)` is `123.45` .
DOUBLE	Eight-byte floating point number	15 decimal digits of precision
DATE	Date	Example: `DATE'1969-07-20'`
TIME	Time	Example: `TIME '20:17:40'`
TIMESTAMP	Timestamp containing both date and time parts	Example: `TIMESTAMP '1969-07-20 20:17:40'`
VARBINARY	Binary data	Byte array

Type conversion

Convertible	Data Type										
	VARCHAR	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	DATE	TIMESTAMP	TIME
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
BOOLEAN	Y	Y	N	N	N	N	N	N	N	N	N
TINYINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
SMALLINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
INT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
BIGINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
FLOAT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DOUBLE	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DATE	Y	N	N	N	N	N	N	N	Y	Y	N
TIMESTAMP	Y	N	N	N	N	N	N	N	Y	Y	Y
TIME	Y	N	N	N	N	N	N	N	N	Y	Y

Type conversion example

Input example

var1 (VARCHAR)	big1 (BIGINT)
1000	323

Sample code

```
CAST(var1 AS BIGINT) AS AA;
CAST(big1 AS VARCHAR) AS BB;
```

Output example

AA (BIGINT)	BB (VARCHAR)
1000	323

3.1.4.2. Data types and operators

This topic describes data types and operators in Realtime Compute.

You can perform mathematical and logical operations for different data types. Examples are provided in the following table.

Operator	Description	Data types supported by numeric1 and numeric2	Example
numeric1 + numeric2	Addition		2 + 4.2
numeric1 - numeric2	Subtraction		3 - 5.3

Operator	Description	Data types supported by numeric1 and numeric2	Example
numeric1 * numeric2	Multiplication	INT, DOUBLE, DECIMAL, BIGINT	2 * 4
numeric1/numeric2	Division		2.4/5
numeric1 > numeric2	>		2.4 > 5
numeric1 < numeric2	<		2.4 < 5
numeric1 >= numeric2	>=		2.4 >= 5
numeric1 <= numeric2	<=		2.4 <= 5
numeric1 = numeric2	=	INT, DOUBLE, DECIMAL, BIGINT, VARCHAR	'iphone' = 5
numeric1 <> numeric2	<>		'iphone' <> 5

3.1.5. DDL statements

3.1.5.1. Overview

Syntax

```
CREATE TABLE tableName
  (columnName dataType [, columnName dataType ]*)
  [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

Description

Realtime Compute does not store data from source and result tables. You can only perform operations on source and result tables by using DDL statements to declare references to data stores. Example:

```
create table tt_stream(
  a varchar,
  b varchar,
  c varchar
  ) with (
  type='DataHub',
  topic='blink_tt_test',
  accessId='yourAccessId',
  accessKey='yourAccessSecret'
  );
```

This Flink SQL statement is not used to create a DataHub topic, but to create a table named tt_stream in Realtime Compute and use the table to declare a reference to a DataHub topic. You can use tt_stream as an alias in subsequent DML statements to perform operations on the table.

- A declaration of a table (tt_stream in this example) is only valid for the jobs generated from the current SQL file. A job is generated after a SQL file is published. The table can also be declared for reference in other SQL files under the same project.
- Since Realtime Compute adopts standard SQL, keywords, table names, and field names in DDL statements are case-insensitive.
- Table and field names must start with a letter or number and can only contain letters, numbers, and underscores (_).
- The method of mapping fields between a source table and an upstream data store depends on the feature of the data store. If the data store such as DataHub supports mapping based on key values, the field names in the source table must be the same as those in the data store. However, the number of fields can be different. If a data store does not support mapping based on key values, both the number and sequence of fields defined in the source table must the same as those in the data store. We recommend that you use the same field names and the same number of fields in DDL statements as those in external data stores. This can prevent data errors caused by inaccurate definitions.

Field mapping

Data stores are classified into two types based on whether the data store has a schema.

- Sequence mapping

  Systems such as Message Queue (MQ) are non-structured storage systems that do not have a schema. You can customize field names in DDL statements. However, the data types and the number of fields must be the same as those in the data store.

  The following example shows an MQ record:

  ```
  asavfa,sddd32,sdfdsv
  ```

  You need to set field names of the tt_stream table based on naming conventions.

```
create table tt_stream(
    a varchar,
    b varchar,
    c varchar
    ) with (
    type = 'mq',
    topic='blink_mq_test',
    accessId='yourAccessId',
    accessKey='yourAccessSecret'
    );
```

- Name mapping

  For structured storage systems such as DataHub, field names and data types are defined based on a table schema. We recommend that you define fields by using Flink SQL based on the table schema. The names, number, and sequence of fields must be the same as those in the data store.

  > **Note** If field names in the data store (such as Table Store) are case-sensitive, you need to enclose field names in backticks (`) in Flink SQL statements. Take DataHub as an example. In DDL statements, field names of the declared source table must apply the same casing that DataHub uses.

  The following table describes a sample schema of DataHub.

  ### A sample schema of DataHub

Field	Data type
name	varchar
age	bigint
value	varchar

  We recommend that you declare references for all fields. Note that you can only reference the fields that are included in the data store. A sample of the DDL statement is provided as follows:

```
create table stream_result (
  name varchar,
  age bigint,
  value varchar
  ) with (
  type='DataHub',
  endpoint='http://dh-cn-hangzhou.aliyuncs.com',
  accessId='yourAccessId',
  accessKey='yourAccessSecret',
  project='project',
  topic = 'topic'
  );
```

Case insensitivity

Realtime Compute adopts standard SQL. Therefore, fields are case-insensitive. For example, the following two statements produce the same effect. Statement 1:

```
create table stream_result (
  name    varchar,
  value   varchar
  );
```

Statement 2:

```
create table STREAM_RESULT (
  NAME    varchar,
  VALUE   varchar
  );
```

Realtime Compute references data from multiple data stores. Some data stores, such as Table Store, are case-sensitive in field names. If a NAME parameter is defined in Table Store, the DDL statement must be as follows:

```
create table STREAM_RESULT (
  `NAME`    varchar,
  `VALUE`   varchar
  );
```

In all subsequent DML statements, the parameter must be enclosed in backticks (`) if it is referenced. A sample code is provided as follows:

```
INSERT INTO Table_A
  SELECT
  `NAME`,
  `VALUE`
  FROM
  Table_B;
```

3.1.5.2. Create a data source table

3.1.5.2.1. Overview

Source tables for Realtime Compute store streaming data. Streaming data triggers stream processing jobs in Realtime Compute. To perform stream processing, you must create at least one source table that provides streaming data for Realtime Compute jobs.

Syntax

```
 CREATE TABLE tableName
   (columnName dataType [, columnName dataType ]*)
   [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

Examples

```
CREATE TABLE metaq_stream(
 x VARCHAR,
 y VARCHAR,
 z VARCHAR
) WITH (
 type='mq',
 topic='<yourTopicName>',
 endpoint='<yourEndpointName>',
 pullIntervalMs='1000',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 startMessageOffset='1000',
 consumerGroup='<yourConsumerGroup>',
 fieldDelimiter='|'
);
```

Retrieve attribute fields from a source table

- Syntax

You must add the `HEADER` keyword to the end of a field declaration in the DDL statement of the source table before you can retrieve attribute fields from a source table.

```
CREATE TABLE sourcetable
(
 `timestamp` VARCHAR HEADER,
 name     VARCHAR,
 MsgID    VARCHAR
)WITH(
   type='RDS'
);
```

In this example, the `` `timestamp` `` field is defined as `HEADER`. Realtime Compute reads the values of the attribute field from the source table, and then uses the field as a general field.

> **Note** The attribute field varies according to the source table type, such as DataHub, Log Service, and Message Queue. You can customize attribute fields for certain types of source tables. For more information, see the topics about source tables.

- Example

  A Log Service source table is used as an example to explain how to retrieve attribute fields. Log Service supports the following three attribute fields.

Field name	Description
`__source__`	The log source
`__topic__`	The log topic
`__timestamp__`	The time when a data record enters Log Service

> **Note** You must add the `HEADER` keyword to the end of a field declaration before you can retrieve attribute fields from a source table.

  Examples

  - Input example

```
__topic__: ens_altar_flow
    result: {"MsgID":"ems0a","Version":"0.0.1"}
```

- Sample code

```
CREATE TABLE sls_log (
 __topic__ VARCHAR HEADER,
 result   VARCHAR
)WITH(
 type ='sls'
);
CREATE TABLE sls_out (
 name   varchar,
 MsgID   varchar,
 Version  varchar
)WITH(
 type ='RDS'
);
INSERT INTO sls_out
SELECT
__topic__,
JSON_VALUE(result,'$.MsgID'),
JSON_VALUE(result,'$.Version')
FROM
sls_log
```

- Output example

name (VARCHAT)	MsgID (VARCHAT)	Version (VARCHAT)
ens_altar_flow	ems0a	0.0.1

Source tables for window function-based computing

Realtime Compute can aggregate data by using window functions based on the event time or processing time. If window functions are used for a stream processing job, you must define the watermark and computed column in the DDL statement of a source table. For more information about the watermark and computed column, see Watermark and Computed column. For information about data aggregation based on time attributes, see Time attributes.

3.1.5.2.2. Create a DataHub source table

This topic describes how to create a DataHub source table for Realtime Compute. It introduces attribute fields, parameters in the WITH clause, and type mapping.

Introduction to DataHub

DataHub is a real-time data distribution platform that is designed to process streaming data. It transmits big data to other Alibaba Cloud Services. DataHub can work with multiple Alibaba Cloud services to build an end-to-end data processing platform. Realtime Compute uses DataHub to store source tables and result tables for stream processing jobs. Data Transmission Services (DTS) and the Internet of Things (IoT) also use DataHub to access big data platforms. DataHub stores streaming data that can be used as input data for Realtime Compute.

Example

You can use the following sample code to create a DataHub source table:

```
create table datahub_stream(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
) with (
  type='datahub',
  endPoint='yourEndpoint',
  project='yourProjectName',
  topic = 'yourTopicName',
  accessId='yourAccessId',
  accessKey='yourAccessSecret',
  startTime='2017-07-21 00:00:00'
);
```

Attribute field (timestamp)

You can use Flink SQL to retrieve the timestamp field of a DataHub source table. You can obtain the system time of each data record by reading the timestamp field of a DataHub source table.

Field name	Description
timestamp	Specifies the system time at which data is written into a DataHub source table.

Attribute field instructions

To extract data from the attribute field of DataHub, you need to add the `HEADER` keyword to the end of a field declaration.

Attribute fields are not defined in the DataHub topic. To extract the `system time` of each data record, you can define a `timestamp` field, and add `HEADER` to the end of the field declaration. Example:

- Input example

name (VARCHAR)	MsgID (VARCHAR)
ens_altar_flow	ems0a

- Sample code

```
CREATE TABLE datahub_log (
 `timestamp` varchar HEADER,
 name     varchar,
 MsgID     varchar
)
WITH
(
 type ='datahub'
);
CREATE TABLE RDS_out (
 `timestamp`    varchar,
 MsgID   varchar,
 name  varchar
)
WITH
(
 type ='RDS'
);
INSERT INTO RDS_out
SELECT
`timestamp`,
MsgID,
name
FROM datahub_log;
```

- Output example

TIME (VARCHAR)	MsgID (VARCHAR)	name (VARCHAR)
1522652455625	ems0a	ens_altar_flow

Parameters in the WITH clause

Currently, topics only support the data records of the tuple pattern.

Parameter	Description	Remarks
endPoint	Specifies the consumer endpoint.	For more information, see *Obtain an endpoint* in *DataHub Developer Guide*.
accessId	Specifies an AccessKey ID.	N/A
accessKey	Specifies an AccessKey secret.	N/A

Parameter	Description	Remarks
project	Specifies a project in DataHub.	N/A
topic	Specifies a topic of the project.	N/A
startTime	Specifies a start offset.	The format is `yyyy-MM-dd hh:mm:ss`.
maxRetryTimes	Specifies a maximum number of read attempts.	Optional. The default value is 20.
retryIntervalMs	Specifies an interval of retries.	Optional. The default value is 1000.
batchReadSize	Specifies the number of data records that are read at a time.	Optional. The default value is 10, and the maximum value is 1000.
lengthCheck	Specifies a rule for checking the number of fields parsed from a row of data.	Optional. The default value is NONE, indicating: • When the number of fields parsed from a row of data is greater than the number of defined fields, data is extracted from left to right based on the order of defined fields. • When the number of fields parsed from a row of data is smaller than the number of defined fields, this row of data is skipped. Other valid values are SKIP, EXCEPTION, and PAD. • SKIP: When the number of fields parsed from data does not match the number of defined fields, this row of data is skipped. • EXCEPTION: When the number of fields parsed from data does not match the number of defined fields, an exception is reported. • PAD: Data is extracted from left to right based on the order of defined fields. ◦ When the number of fields parsed from a row of data is greater than the number of defined fields, data is extracted from left to right based on the order of defined fields. ◦ When the number of fields parsed from a row of data is smaller than the number of defined fields, the values of the missing fields are null.

Parameter	Description	Remarks
columnErrorDebug	Specifies whether debugging is enabled. If you enable debugging, parsing exceptions are logged when they occur.	Optional. The default value is false.

Type mapping

The following table lists the mapping between DataHub and Realtime Compute data types. We recommend that you declare the type mapping in the DDL statement.

DataHub data type	Realtime Compute data type
BIGINT	BIGINT
STRING	VARCHAR
DOUBLE	DOUBLE
TIMESTAMP	BIGINT
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL

Note The TIMESTAMP type field defined in DataHub is a 16-digit Unix timestamp measured in microseconds. In contrast, the TIMESTAMP type field defined in Realtime Compute is a 13-digit Unix timestamp measured in milliseconds. Therefore, we recommend that you convert the TIMESTAMP data type to BIGINT. If you want to use the TIMESTAMP data type in Realtime Compute, we recommend that you use Computed column to convert the time format.

3.1.5.2.3. Create a Log Service source table

This topic describes how to create a Log Service source table for Realtime Compute. It introduces supported attribute fields, parameters in the WITH clause, and type mapping.

Introduction to Log Service

Log Service is a one-stop logging service developed by Alibaba Cloud. It has shown superior performance in big data scenarios. Log Service is used to store streaming data. The streaming data can be used as input data for Realtime Compute. The data format of logs is JSON. An example is described as follows:

```
{
  "a": 1000,
  "b": 1234,
  "c": "li"
}
```

You can define a DDL statement as follows (sls in the code represents Log Service):

```
create table sls_stream(
 a int,
 b int,
 c VARCHAR
) with (
 type ='sls',
 endPoint='yourEndpoint',
 accessId ='yourAccessId',
 accessKey ='yourAccessKey',
 startTime = 'yourStartTime',
 project='yourProjectName',
 logStore='yourLogStoreName',
 consumerGroup ='yourConsumerGroupName'
);
```

Supported attribute fields

Currently, Flink SQL supports three types of Log Service attribute fields by default. Custom fields are also supported.

Field name	Description
`__source__`	Specifies a log source.
`__topic__`	Specifies a log topic.
`__timestamp__`	Specifies the time when a logged event occurs.

Attribute field instructions

To extract data from the supported fields, you need to add the `HEADER` keyword after the data type in the declaration. An example is described as follows:

- Input example

```
__topic__: ens_altar_flow
 result: {"MsgID":"ems0a","Version":"0.0.1"}
```

- Sample code

```
CREATE TABLE sls_log (
 __topic__ VARCHAR HEADER,
 result    VARCHAR
)
WITH
(
 type ='sls',
);
CREATE TABLE sls_out (
 name    VARCHAR,
 MsgID   VARCHAR,
 Version VARCHAR
)
WITH
(
 type ='RDS'
);
INSERT INTO sls_out
SELECT
__topic__
JSON_VALUE (result, '$. MsgID '),
JSON_VALUE (result, '$. version ')
FROM
sls_log
```

- Output example

name (VARCHAR)	MsgID (VARCHAR)	Version (VARCHAR)
ens_altar_flow	ems0a	0.0.1

Parameters in the WITH clause

Parameter	Description	Remarks
endPoint	Specifies the consumer endpoint.	Log Service Endpoint, for example `http://regionid.example.com`.
accessId	Specifies an AccessKey ID.	N/A
accessKey	Specifies an AccessKey Secret.	N/A
project	Specifies a project in Log Service.	N/A

Parameter	Description	Remarks
logStore	Specifies the name of a Logstore in the project.	N/A
consumerGroup	Specifies the name of a consumer group.	You can specify a consumer group name. The format of the consumer group name is not fixed.
startTime	Specifies the time at which a log starts to be consumed.	N/A
heartBeatIntervalMills	Optional. Specifies a heartbeat interval of a consumption client.	The default value is 10 seconds.
maxRetryTimes	Specifies a maximum number of read attempts.	Optional. The default value is 5.
batchGetSize	Specifies the number of log groups that are read at a time.	Optional. The default value is 10.

Parameter	Description	Remarks
lengthCheck	Specifies a rule for checking the number of fields parsed from a row of data.	Optional. The default value is NONE, indicating: • When the number of fields parsed from a row of data is greater than the number of defined fields, data is extracted from left to right based on the order of defined fields. • When the number of fields parsed from a row of data is smaller than the number of defined fields, this row of data is skipped. Other optional values are SKIP, EXCEPTION, and PAD. • SKIP: When the number of fields parsed from data does not match the number of defined fields, this row of data is skipped. • EXCEPTION: When the number of fields parsed from data does not match the number of defined fields, an exception is reported. • PAD: Data is extracted from left to right based on the order of defined fields. ◦ When the number of fields parsed from a row of data is greater than the number of defined fields, data is extracted from left to right based on the order of defined fields. ◦ When the number of fields parsed from a row of data is smaller than the number of defined fields, null is used to fill in the missing fields.
columnErrorDebug	Specifies whether debugging is enabled.	Optional. The default value is false. If you enable debugging, a log entry that indicates a parsing exception is printed.
directMode	Specifies whether the direct connection mode to access Log Service is enabled.	Optional. The default value is false, indicating that the direct connection mode is not enabled. After the direct mode is enabled, you can access Log Service through an Nginx server. An Nginx server demonstrates high performance in handling connection requests.

Note

- Currently, Log Service does not support the MAP data type.
- We recommend that you define the fields in the same order as the fields in the preceding table. Unordered fields are also supported.
- If input data is in the JSON format, define a delimiter and use the built-in function JSON_VALUE to analyze the data. Otherwise, a parsing error will occur and the following error message will appear:

```
2017-12-25 15:24:43,467 WARN [Topology-0 (1/1)] com.alibaba.blink.streaming.connectors.com
mon.source.parse.DefaultSourceCollector - Field missing error, table column number: 3, data co
lumn number: 3, data filed number: 1, data: [{"lg_order_code":"LP00000005","activity_code":"T
EST_CODE1","occur_time":"2017-12-10 00:00:01"}]
```

- The batchGetSize value must not exceed 1000. Otherwise, an error will occur.
- The batchGetSize parameter specifies the number of log groups that are read at a time. If the size of a single log entry and the batchGetSize value are too large, frequent garbage collection is generated. To avoid this issue, you need to set this parameter to a small value.

Type mapping

The following table describes the mapping between Log Service data types and Realtime Compute data types. We recommend that you declare the mapping in a DDL statement.

Log Service data type	Realtime Compute data type
STRING	VARCHAR

3.1.5.3. Create a data result table

3.1.5.3.1. Overview

Realtime Compute uses the `CREATE TABLE` statement to define the schema of a result table for output data and how data is written into a target data store.

Data can be written by using either of the following methods: append and update.

- Append: If the result table is stored in a relational database with no primary key defined, Log Service, or Message Queue, output data is appended to the result table.
- Update: If the result table is stored in a relational database or a Hadoop database with a primary key defined, output data is written into the result table as follows:
  - If the value for the primary key is not found in the result table, the data record is inserted into the database.
  - If the value for the primary key is found, the existing data record in the database is overwritten.

Result table syntax

```
CREATE TABLE tableName
  (columnName dataType [, columnName dataType ]*)
  [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

Example

```
CREATE TABLE rds_output(
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='rds',
url='yourDatabaseURL',
tableName='yourTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);
```

3.1.5.3.2. Create a DataHub result table

This topic describes how to create a DataHub result table for Realtime Compute.

Introduction to DataHub

DataHub is a data distribution platform that is designed to process streaming data. It provides a channel for other Apsara Stack services to process big data. DataHub works with multiple Alibaba Cloud services to provide an end-to-end data processing solution. In Realtime Compute, you can use DataHub to store input and output data.

DDL syntax

In Realtime Compute, you can create a DataHub result table to store output data. The sample code is provided as follows:

```
create table datahub_output(
 name varchar,
 age BIGINT,
 birthday BIGINT
)with(
 type='datahub',
 endPoint='yourEndpoint',
 project='yourProjectName',
 topic='yourTopicName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret',
 batchSize='20',
 batchWriteTimeoutMs='1000'
);
```

Parameters in the WITH clause

Parameter	Description	Remarks
endPoint	Specifies the endpoint.	For more information, see *Obtain an endpoint* in *DataHub Developer Guide*.
project	Specifies the name of a DataHub project.	N/A
topic	Specifies the name of a DataHub topic.	N/A
accessId	Specifies the AccessKey ID.	N/A
accessKey	Specifies the AccessKey Secret.	N/A
maxRetryTimes	Specifies the maximum number of retries for writing data to the result table.	Optional. The default value is 3.
batchSize	Specifies the number of data records that are written at a time.	Optional. The default value is 300.
batchWriteTimeoutMs	Specifies the timeout period for writing data.	Optional. The default value is 5000.
maxBlockMessages	Specifies the maximum number of data blocks that are written at a time.	Optional. The default value is 100.

3.1.5.3.3. Create a Log Service result table

This topic describes how to create a Log Service result table for Realtime Compute.

Introduction to Log Service

Log Service (LOG) provides an end-to-end management solution for real-time log data. Log Service allows you to quickly collect, consume, ship, query, and analyze log data without the need for further development. Log Service has high performance in data O&M. You can use Log Service to process massive logs to meet business demands in the data technology (DT) era.

DDL syntax

In Realtime Compute, you can use Log Service to store output data. The sample code is provided as follows:

```
create table sls_stream(
 name varchar,
 age BIGINT,
 birthday BIGINT
)with(
 type='sls',
 endPoint='yourEndpoint',
 accessId='yourAccessId',
 accessKey='yourAccessKey',
 project='yourProjectName',
 logStore='yourLogStoreName'
);
```

Parameters in the WITH clause

Parameter	Description	Remarks
endPoint	Specifies the endpoint.	Log Service Endpoint, for example `http://regionid.example.com`.
project	Specifies the name of a Log Service project.	N/A
topic	Specifies the name of a Log Service topic.	N/A
accessId	Specifies the AccessKey ID.	N/A
accessKey	Specifies the AccessKey Secret.	N/A

Parameter	Description	Remarks
mode	Specifies the mode for writing data to the result table.	Optional. The default value is `random`. If you set this parameter to `partition`, the result table is partitioned and data is written to the specified partition.
partitionColumn	Specifies the partition column.	This parameter is required if the `mode` parameter is set to `partition`.
source	Specifies the source of logs. For example, you can set this parameter to the IP address of the server where logs are generated.	Optional. The default value is NONE.

3.1.5.3.4. Create a Table Store result table

This topic describes how to create a Table Store result table for Realtime Compute. This topic also introduces the mapping between Table Store data types and Realtime Compute data types.

Introduction to Table Store

Table Store is a distributed NoSQL data storage service developed based on the Apsara distributed system. Based on data sharding and load balancing technologies, Table Store has high performance in scaling out and handling concurrent transactions. You can use Table Store to store and query large amounts of structured data.

DDL syntax

In Realtime Compute, you can use Table Store to store output data. The sample code is provided as follows:

```
CREATE TABLE stream_test_hotline_agent (
 name varchar,
 age BIGINT,
 birthday BIGINT,
 primary key(name,age)
) WITH (
 type='ots',
 instanceName='yourInstanceName',
 tableName='yourTableName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret',
 endPoint='yourEndpoint',
 valueColumns='birthday'
);
```

Note The value of the valueColumns parameter can be the name of any field other than a declared primary key.

Parameters in the WITH clause

Parameter	Description	Remarks
instanceName	Specifies the name of a Table Store instance.	N/A
tableName	Specifies the name of a result table.	N/A
endPoint	Specifies the endpoint to access the instance.	Table Store Endpoint, for example `http://regionid.example.com`.
accessId	Specifies the AccessKey ID.	N/A
accessKey	Specifies the AccessKey Secret.	N/A
valueColumns	Specifies the names of fields to be inserted into the result table. Multiple fields are separated with commas (`,`).	For example, you can insert two fields in the following format: `'ID,NAME'`.
bufferSize	Specifies the buffer size after data deduplication.	Optional. The default value is 5000, indicating that data output is triggered after 5,000 input data records are processed.

Parameter	Description	Remarks
batchWriteTimeoutMs	Specifies the timeout period for writing data.	Optional. The default value is 5000, measured in milliseconds. This value indicates that if no data is written to Table Store within 5,000 milliseconds (5 seconds), all cached data is written to the result table.
batchSize	Specifies the number of data records that are written at a time.	Optional. The default value is 100.
retryIntervalMs	Specifies the retry interval.	Optional. The default value is 1000, measured in milliseconds.
maxRetryTimes	Specifies the maximum number of retries for writing data to the result table.	Optional. The default value is 100.
ignoreDelete	Specifies whether to skip delete operations.	The default value is false.

Type mapping

Table Store data type	Realtime Compute data type
INTEGER	BIGINT
STRING	VARCHAR
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

Note You must define a `primary key` in a Table Store result table. Output data is appended to the Table Store result table to update the result.

3.1.5.3.5. Create an RDS result table

This topic describes how to create a Relational Database Service (RDS) result table for Realtime Compute. This topic also introduces the mapping between Realtime Compute data types and RDS data types.

Introduction to RDS

ApsaraDB for Relational Database Service (RDS) offers stable, reliable, and scalable online database services. Based on the distributed file system and high-performance storage system developed by Alibaba Cloud, RDS supports a variety of database engines such as MySQL, SQL Server, and PostgreSQL. RDS also supports Postgre Plus Advanced Server (PPAS) that is highly compatible with Oracle databases. RDS provides comprehensive solutions for database operations and management (O&M), including disaster recovery, backup, monitoring, and migration.

> **Note** An RDS result table is consistent with the parameters in the WITH clause. When you create an RDS result table, the RDS data store must have the corresponding table stored in the database.

DDL syntax

In Realtime Compute, you can use RDS to store output data. Currently, only RDS for MySQL is supported. The sample code is provided as follows:

```
create table rds_output(
 id int,
 len int,
 content varchar,
 primary key(id,len)
) with (
 type='rds',
 url='yourDatabaseURL',
 tableName='yourDatabaseTableName',
 userName='yourDatabaseUserName',
 password='yourDatabasePassword'
);
```

> **Note**
>
> - In Realtime Compute, each row of output data is converted into a row of strings. The strings are then written into an RDS result table. If you want to write multiple rows of output data at a time, you need to add `?rewriteBatchedStatements=true` to the end of the URL. This ensures optimal efficiency when writing multiple rows.
> - For a result table stored in an RDS for MySQL instance, you can define an auto-increment primary key. If you want to use the auto-increment primary key in the result table, do not declare the auto-increment field in the DDL statement. For example, if ID is an auto-increment field in RDS for MySQL, do not declare the ID field in the DDL statement. When writing a row of output data to the database, a value is automatically generated for the auto-increment field.

Parameters in the WITH clause

Parameter	Description	Remarks
url	Specifies the URL for RDS.	For more information, see *Connect to a MySQL instance from a client* in *RDS for MySQL User Guide*.
tableName	Specifies the name of a result table.	N/A
userName	Specifies the username that is used to access RDS.	N/A
password	Specifies the password that is used to access RDS.	N/A
maxRetryTimes	Specifies the maximum number of retries for writing data to the result table.	Optional. The default value is 3.
batchSize	Specifies the number of data records that are written at a time.	Optional. The default value is 50.
bufferSize	Specifies the buffer size after data deduplication. You can only use this parameter when a primary key is defined.	Optional. The default value is 1, indicating that when one data record is processed, an output is generated. The default value is increased to 1000 in Realtime Compute version 1.4.1.
flushIntervalMs	Specifies the timeout period of writing data.	Optional. The unit of measurement is milliseconds. The default value is 5000, indicating that if no data is written within 5,000 milliseconds (5 seconds), all cached data is written to the result table.
excludeUpdateColumns	Specifies one or more columns to be excluded when you update the result table based on key values.	Optional. The default value is NONE, indicating that only primary key fields are excluded.
ignoreDelete	Specifies whether to skip delete operations.	The default value is false.
partitionBy	Specifies the partitioning rule for the result table. Before data is written into the result table, Hash partitioning is performed based on a partition key. Data flows to the corresponding node defined by the hash function.	Optional. The default value is NONE.

Type mapping

RDS data type	Realtime Compute data type
TEXT	VARCHAR
BYTE	VARCHAR
INTGER	INT
LONG	BIGINT
DOUBLE	DOUBLE
DATE	VARCHAR
DATETIME	VARCHAR
TIMESTAMP	VARCHAR
TIME	VARCHAR
YEAR	VARCHAR
FLOAT	FLOAT
DECIMAL	DECIMAL
CHAR	VARCHAR

JDBC connection parameters

Parameter	Description	Default value	Since version (JDBC driver)
useUnicode	Specifies whether to use the Unicode character set. If you want to set the characterEncoding parameter to gb2312 or gbk, this parameter must be set to true.	false	1.1g
characterEncoding	Specifies the character encoding when the useUnicode parameter is set to true. You can set this parameter to gb2312 or gbk.	false	1.1g

Parameter	Description	Default value	Since version (JDBC driver)
autoReconnect	Specifies whether to automatically re-establish a connection when the connection to the database is unexpectedly interrupted.	false	1.1
autoReconnectForPools	Specifies whether to use the reconnection policy for a database connection pool.	false	3.1.3
failOverReadOnly	Specifies whether to set the connection to read-only after the database is automatically reconnected.	true	3.0.12
maxReconnects	Specifies the maximum number of connection retries when the autoReconnect parameter is set to true.	3	1.1
initialTimeout	Specifies the interval between two connection retries when the autoReconnect parameter is set to true. The unit of measurement is seconds.	2	1.1
connectTimeout	Specifies the timeout period when you use a socket connection to access the database server. The unit of measurement is seconds. The default value of 0 indicates no timeout (only works on JDK version 1.4 and later).	0	3.0.1

Parameter	Description	Default value	Since version (JDBC driver)
socketTimeout	Specifies the timeout period for a socket-based read/write operation. The unit of measurement is seconds. The value of 0 indicates no timeout.	0	3.0.1

FAQ

- Q: When output data is written to an RDS table, is a new data record generated in the table? If not, is the result table updated based on the primary key value?

  A: If a primary key is defined in the DDL statement, the result table is updated in the `insert into on duplicate key update` method. For a data record, if the value of primary key field does not exist, the record is inserted into the table as a new row. If the value of primary key field exists, the original row of data in the table is updated. If a primary key is not declared in the DDL statement, output data is added to the result table by using the `insert into` method.

- Q: How can I perform GROUP BY operations based on the unique index of an RDS table?

  A: An RDS table has only one auto-increment field, which cannot be declared as the primary key in the Realtime Compute DDL statement. If you want to perform GROUP BY operations based on the unique index of an RDS table, declare the unique index in primary key() in the DDL statement.

3.1.5.4. Create a data dimension table

3.1.5.4.1. Overview

You must add `PERIOD FOR SYSTEM_TIME` in the CREATE TABLE statement that is used to create a dimension table. `PERIOD FOR SYSTEM_TIME` defines the period of time when a data record is valid. This means that the table is frequently updated.

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id), -- You must define a primary key for the dimension table.
  PERIOD FOR SYSTEM_TIME -- Defines the period of time when a data record is valid.
  ) with (
  type = 'rds',
  ...
  )
```

Note

- When you declare a dimension table, you must specify the primary key.
- When you join a dimension table with another table, the ON condition must contain an equivalence condition that includes the primary key of either table.

3.1.5.4.2. Create a Table Store dimension table

This topic describes how to create a Table Store dimension table for Realtime Compute.

Table Store is a distributed NoSQL data storage service developed based on the Apsara distributed system. Based on data sharding and load balancing technologies, Table Store has high performance in scaling out and handling concurrent transactions. You can use Table Store to store and query large amounts of structured data.

DDL syntax

```
CREATE TABLE ots_dim_table (
 id int,
 len int,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME -- Defines the period of time when a data record is valid. This means that the tabl
e is frequently updated.
) WITH (
 type='ots',
 endPoint='yourEndpoint'
 instanceName='yourInstanceName',
 tableName='yourTableName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret',
);
```

Note When you declare a dimension table, you must specify the primary key. When you join a dimension table with another table, the ON condition must contain an equivalent condition that includes the primary key of either table. The primary key of a Table Store table is the rowkey.

Parameters in the WITH clause

Parameter	Description
instanceName	Specifies the name of a Table Store instance.

Parameter	Description
tableName	Specifies the name of a table in the Table Store instance.
endPoint	Specifies the endpoint that is used to access the instance.
accessId	Specifies the AccessKey ID.
accessKey	Specifies the AccessKey Secret.

Cache parameters

Parameter	Description	Remarks
cache	Specifies the cache strategy.	The default value is None. You can set the parameter to LRU.
cacheSize	Specifies the cache size.	Specifies the maximum number of data records that can be cached. You can specify this parameter if you set the cache parameter to LRU. The default value is 10000.
cacheTTLMs	Specifies the time-to-live (TTL) of each data record that is cached. The TTL is measured in milliseconds.	You can specify this parameter if you set the cache parameter to LRU. No TTL is set by default.

Sample code

```
CREATE TABLE datahub_input1 (
id        BIGINT,
name      VARCHAR,
age       BIGINT
) WITH (
type='datahub'
);
CREATE TABLE phoneNumber(
name VARCHAR,
phoneNumber BIGINT,
PRIMARY KEY(name),
PERIOD FOR SYSTEM_TIME -- Defines the period of time when a data record is valid.
)WITH(
type='ots'
);
CREATE table result_infor(
id BIGINT,
phoneNumber BIGINT,
name VARCHAR
)WITH(
type='rds'
);
INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w -- You must include this clause in the INSERT
INTO statement if you are performing a JOIN query for a dimension table.
ON t.name = w.name;
```

3.1.5.4.3. Create an RDS dimension table

This topic describes how to create an ApsaraDB for RDS dimension table in Realtime Compute.

ApsaraDB for RDS offers stable, reliable, and scalable online database services. Based on the distributed file system and high-performance storage services of Apsara Stack, RDS supports a variety of database engines such as MySQL, SQL Server, PostgreSQL, and Postgre Plus Advanced Server (PPAS). PPAS is compatible with Oracle databases. RDS provides a wide range of solutions for database operations and management (O&M), including disaster recovery, backup, restoration, monitoring, and migration.

DDL syntax

RDS can be used to store dimension tables. Only ApsaraDB RDS for MySQL is supported.

```
CREATE TABLE rds_dim_table(
 id int,
 len int,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME--Indicates that a dimension table is used.
) with (
 type='rds',
 url='rdsDatabaseURL',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

Note When you declare a dimension table, you must specify the primary key. You can join a dimension table with another table by using a JOIN statement. All the primary key fields of the dimension table must be on the left or right side of the equal operator in the ON clause of the JOIN statement. The primary key of an RDS dimension table can be used as the primary key or unique index of the result table.

Parameters in the WITH clause

Parameter	Description	Remarks
url	The database URL	N/A
tableName	The name of the table in the database	N/A
userName	The username that is used to log on to the database	N/A
password	The password that is used to log on to the database	N/A
maxRetryTimes	The maximum number of INSERT attempts	Optional. The default value is 3.

Cache parameters

Parameter	Description	Remarks

Parameter	Description	Remarks
cache	The cache strategy.	The default value is None. Other valid values include LRU and ALL.
cacheSize	The cache size. Unit: lines.	If the cache parameter is set to LRU, you must specify the cache size. The default value is 10000.
cacheTTLMs	The cache time-out period. Unit: milliseconds.	If the cache parameter is set to LRU, this parameter specifies the time before cache entries expire. By default, cache entries do not expire. If the cache parameter is set to ALL, this parameter specifies the cache refresh interval. By default, cache entries are not refreshed.
cacheReloadTimeBlackList	The time periods during which the cache is not refreshed, such as the Double 11. You can specify this parameter only if the cache parameter is set to ALL.	Optional. By default, the parameter is not specified. An example is used to describe the value format: 2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00. To meet format requirements, separate time periods with commas (,), and separate the start and end time of each period with the right arrow sign (->).

> **Note**
>
> RDS provides the following three cache strategies:
>
> - None: indicates that data is not cached.
> - LRU: indicates that the recently used data is cached. If this cache strategy is used, you must specify the cacheSize and cacheTTLMs parameters.
> - ALL: indicates that all data is cached. Before you run a job, all data of the dimension table is loaded into the memory of the operator that runs the JOIN statement. Queries for the dimension table are performed based on the data in the memory. If no result data is found in the memory, no results are returned. All the cached data is refreshed at a specified interval. You can use this cache strategy when a small dimension table is joined with a source table and only a few data records match the condition specified by the ON clause. If this cache strategy is used, you must specify the cacheTTLMs and cacheReloadTimeBlackList parameters.
>
> > **Note**
> >
> > - If you set the cache parameter to ALL, Realtime Compute performs asynchronous cache refresh. Therefore, you must increase the memory space of the operator that runs the JOIN statement. The memory space must be twice the size of the dimension table data.
> > - If you set the cache parameter to ALL, you must monitor the memory space of the operator to prevent out of memory (OOM) errors.

Sample code

```
CREATE TABLE datahub_input1 (
id         BIGINT,
name       VARCHAR,
age        BIGINT
) WITH (
type='datahub'
);
create table phoneNumber(
name VARCHAR,
phoneNumber bigint,
primary key(name),
PERIOD FOR SYSTEM_TIME--Indicates that a dimension table is used.
)with(
type='rds'
);
CREATE table result_infor(
id bigint,
phoneNumber bigint,
name VARCHAR
)with(
type='rds'
);
INSERT INTO result_infor
SELECT
t.id
,w.phoneNumber
,t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w--You must add FOR SYSTEM_TIME AS OF PRO
CTIME() to the end of the dimension table name if you are performing a JOIN operation for the dimension ta
ble.
ON t.name = w.name;
```

3.1.6. DML statements

3.1.6.1. INSERT INTO statement

This topic describes how to use the INSERT INTO statement in Realtime Compute.

Syntax

```
INSERT INTO tableName
    [ (columnName[ , columnName]*) ]
      queryStatement;
```

Example

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
INSERT INTO Orders(z, v)
SELECT c,d FROM OO;
```

- A Realtime Compute job created from an SQL file can include multiple DML operations. It also supports multiple data stores for input and output data, and supports multiple dimension tables. For example, a job can contain two paragraphs of SQL statements written for different businesses. You can use SQL statements to write output data into different data stores.
- In Realtime Compute, you cannot use separate SELECT statements. SELECT statements must be used after CREATE VIEW or INSERT INTO statements.
- You can update a table by using the INSERT INTO statement. For example, you can use an INSERT INTO statement to write a data record with a key value into an RDS table. If the key value is found in the table, the existing data record in the table is overwritten. If the key value is not found, the data record is added to the table.

Table types and statements

Table type	Statement
Source table	You can use the FROM clause. The INSERT INTO statement is not supported.
Dimension table	You can use the JOIN statement. The INSERT INTO statement is not supported.
Result table	You can only use the INSERT INTO statement.
View	You can only use the FROM clause.

3.1.7. QUERY statements

3.1.7.1. SELECT statement

You can use the SELECT statement to select data from the table.

Syntax

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

Input example

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

Example 1

```
SELECT * FROM tableName;
```

Output example

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

Example 2

```
SELECT a, c AS d FROM tableName;
```

Input example

a (VARCHAR)	d (DATE)
a1	1990-02-20
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

Example 3

```
SELECT DISTINCT a FROM tableName;
```

Output example

a (VARCHAR)
a1
b1
c1

Subquery

The subquery can be nested in the SELECT statement. In the subquery, the table alias must be used to specify which table reference is to be used. A sample code is provided as follows:

```
INSERT INTO result_table
SELECT * from
    (SELECT  t.a,
          sum(t.b) AS sum_b
         FROM t1 t
         GROUP BY t.a) t1
WHERE t1.sum_b > 100;
```

Output example

a (VARCHAR)	b (INT)
a1	211
b1	120
a1	257

3.1.7.2. WHERE clause

To specify the condition based on which data is selected from the table, you can add the WHERE clause to the SELECT statement.

Syntax

```
SELECT [ ALL | DISTINCT ]
 { * | projectItem [, projectItem ]* }
 FROM tableExpression
 [ WHERE booleanExpression ];
```

The following table describes the operators that can be used in the WHERE clause.

Operators

Operator	Description
=	Equal to
<>	Not equal to
>	More than
>=	More than or equal to
<	Less than
<=	Less than or equal to

Example

Input example

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	Shanghai

```
SELECT * FROM XXXX WHERE City='Beijing'
```

Output example

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

3.1.7.3. HAVING clause

The WHERE clause cannot be used with aggregate functions. To solve this, you can use a HAVING clause instead.

Syntax

```
SELECT [ ALL | DISTINCT ]
 { * | projectItem [, projectItem ]* }
 FROM tableExpression
 [ WHERE booleanExpression ]
 [ GROUP BY { groupItem [, groupItem ]* } ]
 [ HAVING booleanExpression ];
```

Example

Input example

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

```
SELECT Customer,SUM(OrderPrice) FROM XXX
GROUP BY Customer
HAVING SUM(OrderPrice)<2000;
```

Output example

Customer	SUM (OrderPrice)
Carter	1700

3.1.7.4. GROUP BY clause

The GROUP BY clause is used with aggregate functions to split a result set into groups based on one or more columns.

Syntax

```
SELECT [ DISTINCT ]
 { * | projectItem [, projectItem ]* }
 FROM tableExpression
 [ GROUP BY { groupItem [, groupItem ]* } ];
```

Example

Input example

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

```
SELECT Customer,SUM(OrderPrice) FROM XXXX
GROUP BY Customer;
```

Output example

Customer	SUM (OrderPrice)
Bush	2000
Carter	1700
Adams	2000

3.1.7.5. JOIN statement

For JOIN statements, Realtime Compute has the same semantics as BatchCompute. You can use a JOIN statement to join two tables. In Realtime Compute, the two tables are dynamic ones. The result of a JOIN query is dynamically updated. After a specific period, Realtime Compute returns the same result as BatchCompute.

Syntax

```
tableReference [, tableReference ]* | tableexpression
[ LEFT ] JOIN tableexpression [ joinCondition ];
```

> **Note**
>
> - Realtime Compute supports EQUI JOINS. NON EQUI JOINS are not supported.
> - Only INNER JOINS and LEFT OUTER JOINS are supported.
> - The fields and primary key defined in the DDL statement of tables to be joined must exist in the data store.

Example 1

```
SELECT o.rowtime, o.productId, o.orderId, o.units,
  p.name, p.unitPrice
  FROM Orders AS o
  JOIN Products AS p
  ON o.productId = p.productId;
```

Output example

rowtime	productId	orderId	units	name	unitPrice
10:17:00	30	5	4	Cheese	17
10:17:05	10	6	1	Beer	0.25
10:18:05	20	7	2	Wine	6
10:18:07	30	8	20	Cheese	17
11:02:00	10	9	6	Beer	0.25
11:04:00	10	10	1	Beer	0.25
11:09:30	40	11	12	Bread	100
11:24:11	10	12	4	Beer	0.25

Example 2

Input example (table name: datahub_stream1)

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21

Input example (table name: datahub_stream2)

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21
0	10	test31
1	10	test41

Sample code

```
-- Creates a DataHub source table.
create table datahub_stream1 (
 a int,
 b int,
 c varchar
) WITH (
   type='datahub',
   endpoint='yourEndpoint',
   accessId='yourAccessId',
   accessKey='yourAccessSecret',
   projectName='yourProjectName',
   topic='yourTopic',
   project='yourProjectName'
);
-- Creates another DataHub source table.
create table datahub_stream2 (
   a int,
   b int,
   c varchar
) WITH (
   type='datahub',
   endpoint='yourEndpoint',
   accessId='yourAccessId',
   accessKey='yourAccessSecret',
   projectName='yourProjectName',
   topic='yourTopicName',
   project='yourProjectName'
);
-- Creates an RDS result table.
create table rds_output(
    s1_c varchar,
```

```
  s2_c varchar
)with(
  type='rds',
  url='yourDatabaseURL',
  tableName='udf',
  userName='yourDatabaseUserName',
  password='yourDatabasePassword'
);
insert into rds_output
select
  s1.c,s2.c
from datahub_stream1 AS s1
join datahub_stream2 AS s2 on s1.a =s2.a
where s1.a = 0;
```

Output example

s1_c (VARCHAR)	s2_c (VARCHAR)
test1	test11
test1	test31

3.1.7.6. JOIN statement for dimension tables

Introduction

Dimension tables are frequently updated. If you want to join a data record with a dimension table, you must specify the creation time of the table snapshot to be used in the JOIN operation. Currently, you can only use the snapshot that is created at the current system time. In the future, Realtime Compute will allow you to use a snapshot created at the time indicated by a value of the rowtime field.

Syntax

```
SELECT column-names
FROM table1  [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF PROCTIME() [AS <alias2>]
ON table1.column-name1 = table2.key-name1
```

Description

Dimension tables support INNER JOIN and LEFT JOIN. Dimension tables do not support RIGHT JOIN and FULL JOIN. As shown in the preceding statement, when you join a data record with a dimension table, you must include the `FOR SYSTEM_TIME AS OF PROCTIME()` subclause in the JOIN statement. The subclause indicates that the JOIN operation is performed on the table snapshot created at the current system time. After a data record arrives at the left table, the system searches for matching values in the dimension table and returns the search result. If no search result is found but later a new data record with a matching value is inserted to the dimension table, the previous search result is not updated. JOIN queries work on processing time. Any later operations in the dimension table such as deletion do not change the returned result.

An example of joining data records with dimension tables is provided as follows:

```
SELECT e.*, w. *
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id
```

> **Note**
>
> - In the ON clause, you must specify an equivalent condition that includes the primary key of either table. You can also add other conditions in the ON clause. For example, `ON event.id = white_list.id AND event.name =white_list.name`.
> - You cannot perform JOIN queries between dimension tables.
> - The fields and primary key defined in the DDL statement of a left table must exist in the data store.

Example

Input example (table name: nameinfor)

id (BIGINT)	name (VARCHAR)	age (BIGINT)
1	lilei	22
2	hanmeimei	20
3	libai	28

Input example (table name: phoneNumber)

name (VARCHAR)	phoneNumber (BIGINT)
dufu	18867889855
baijuyi	18867889856
libai	18867889857

name (VARCHAR)	phoneNumber (BIGINT)
lilei	18867889858

Sample code

```
CREATE TABLE datahub_input1 (
id        BIGINT,
name      VARCHAR,
age       BIGINT
) WITH (
type='datahub',
);
create table phoneNumber(
name VARCHAR,
phoneNumber BIGINT,
primary key(name),
PERIOD FOR SYSTEM_TIME
)with(
type='rds'
);
CREATE table result_infor(
id BIGINT,
phoneNumber BIGINT,
name VARCHAR
)with(
type='rds'
);
INSERT INTO result_infor
SELECT
t.id
,w.phoneNumber
,t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.name = w.name;
```

Output example

id (BIGINT)	phoneNumber (BIGINT)	name (VARCHAR)
1	18867889858	lilei

id (BIGINT)	phoneNumber (BIGINT)	name (VARCHAR)
3	18867889857	libai

3.1.7.7. UNION ALL clause

You can use the UNION ALL clause to merge multiple data streams.

Syntax

```
select_statement
 UNION ALL
 select_statement;
```

> **Note**
>
> - If you need to merge multiple data streams, the fields of the data streams must have the same data types and sequences.
> - Realtime Compute does not support the UNION clause. The UNION ALL clause returns all merged streaming data records, including duplicates. The UNION statement removes duplicates.

Example 1

```
SELECT *
   FROM (
   (SELECT user FROM tableName WHERE a % 2 = 0)
   UNION ALL
   (SELECT user FROM tableName WHERE b = 0)
   );
```

Example 1

Input example (table name: test_source_union1)

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10

Input example (table name: test_source_union2)

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test2	2	20

Input example (table name: test_source_union3)

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10
test2	2	20
test1	1	10

Sample code

```
create table test_source_union1 (
 a VARCHAR,
 b BIGINT,
 c BIGINT
) WITH (
   type='datahub',
   endpoint='yourEndpoint',
   accessId='yourAccessId',
   accessKey='yourAccessSecret',
   projectName='yourProjectName',
   topic='yourTopicName',
   project='yourProjectName'
);
create table test_source_union2 (
 a VARCHAR,
 b BIGINT,
 c BIGINT
) WITH (
   type='datahub',
   endpoint='yourEndpoint',
   accessId='yourAccessId',
   accessKey='yourAccessSecret',
   projectName='yourProjectName',
   topic='yourTopicName',
   project='yourProjectName'
);
create table test_source_union3 (
 a VARCHAR,
```

```
 b BIGINT,
 c BIGINT
) WITH (
   type='datahub',
   endpoint='yourEndpoint',
   accessId='yourAccessId',
   accessKey='yourAccessSecret',
   projectName='yourProjectName',
   topic='yourTopicName',
   project='yourProjectName'
);
create table test_result_union (
   d VARCHAR,
   e BIGINT,
   f BIGINT,
   primary key(d)
) WITH (
    type='rds',
    url='yourDatabaseURL',
    username='yourUserName',
    password='yourPassword',
    tableName='yourTableName'
);
INSERT into test_result_union
SELECT
   a,
   sum(b),
   sum(c)
FROM
   (SELECT * from test_source_union1
   UNION ALL
   SELECT * from test_source_union2
   UNION ALL
   SELECT * from test_source_union3
   )t
 GROUP BY a;
```

Output example

d (VARCHAR)	e (BIGINT)	f (BIGINT)
test1	1	10
test2	2	20
test1	2	20
test1	3	30
test2	4	40
test1	4	40

3.1.7.8. Top_N statement

You can use the Top_N statement to retrieve the largest or smallest N data records from data streams based on the specified metric. In Flink SQL, you can define an over window to perform Top_N queries.

Syntax

```
SELECT *
FROM (
 SELECT *,
  ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
  ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
 FROM table_name)
WHERE rownum <= N [AND conditions]
```

The parameters are described as follows:

- `ROW_NUMBER()` : specifies an over window to compute the row number. The value of the row number starts from 1.
- `PARTITION BY col1[, col2..]` : optional. Specifies one or more columns based on the data records that are partitioned.
- `ORDER BY col1 [asc|desc][, col2 [asc|desc]...]` : specifies one or more columns based on which the rows in a partition are ordered. You can define different ordering directions for different columns.

As indicated by the syntax, the query includes two steps. In the first step, the `ROW_NUMBER()` function is used to specify an over window to compute the row number for the rows that are ordered in a partition. The second step is to return the top N data records. For example, if N is 10, only the top 10 data records are returned.

When you use Flink SQL to perform Top_N queries, input data records are ordered and ranked based on the specified columns. If the top N data records in a partition change, the changed data records are sent to the downstream data store to update the result. You can only export the top N data records to a result table that is defined with a primary key.

The Top_N statement has specific constraints. To allow Flink SQL to identify a Top_N query, you must specify the `rownum <= N` condition in the WHERE clause. You cannot use an expression such as `rownum - 5 <= N` to specify a condition. If you specify multiple conditions in the WHERE clause, you must use AND to separate the conditions.

Example 1

You can use the following example to return the top 100 keywords that are most queried in a city within an hour. For the result table, you must define the rownum, start_time, and city columns as the composite primary key. The composite primary key can uniquely identify a data record.

```
CREATE TABLE rds_output (
 rownum int,
 start_time bigint,
 city varchar,
 keyword varchar,
 pv bigint,
 PRIMARY KEY (rownum, start_time, city)
) with (
 type = 'rds',
 ...
)
INSERT INTO rds_output
SELECT rownum, start_time, city, keyword, pv
FROM (
 SELECT *,
   ROW_NUMBER() OVER (PARTITION BY start_time, city ORDER BY pv desc) AS rownum
 FROM (
     select substr(time_str,1,12) as start_time,
       keyword,
       count(1) AS pv,
       city
     from tmp_search
     group by substr(time_str,1,12), keyword, city
   ) a
)
WHERE rownum <= 100
```

Example 2

Input example

ip (varchar)	time (varchar)
192.168.1.1	100000000
192.168.1.2	100000000
192.168.1.2	100000000
192.168.1.3	100030000
192.168.1.3	100000000
192.168.1.3	100000000

Sample code

```
create table source_table (
 IP VARCHAR,
 `TIME` VARCHAR
)with(
 type='datahub',
 endPoint='http://dh-cn-hangzhou.aliyuncs.com',
 project='Flink_test',
 topic='yourTopicName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret'
);
create table result_table (
 rownum int,
 start_time VARCHAR,
 IP VARCHAR,
 cc BIGINT,
 PRIMARY KEY (start_time, IP)
) with (
 type = 'rds',
 url='jdbc:mysql://rm-bp1gz4k202t8****s.com:3306/Flink_test',
 tableName='yourTableName',
 userName='yourAccessId',
 password='yourAccessPassword'
);
INSERT INTO result_table
SELECT rownum,start_time,IP,cc
FROM (
 SELECT *,
   ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY cc desc) AS rownum
 FROM (
     SELECT SUBSTRING(`TIME`,1,2) AS start_time, -- The input example of the TIME parameter is provided in the preceding table. You can change the value of the TIME parameter based on your situation.
     COUNT(IP) AS cc,
     IP
     FROM  source_table
     GROUP BY SUBSTRING(`TIME`,1,2), IP
   )a
)
WHERE rownum <= 3 -- You can change the number 3 based on your situation.
```

Output example

rownum (INT)	start_time (VARCHAR)	ip (VARCHAR)	cc (BIGINT)
1	10	192.168.1.3	6
2	10	192.168.1.2	4
3	10	192.168.1.1	2

No ranking number optimization

As indicated by the syntax, the rownum field is written to the result table as one field of the primary key. Many repeated data records may be written to the result table. For example, if a data record with the ninth rank is updated and its rank is changed to the first, all top 9 data records are updated to the result table. During the process, workloads increase significantly for the result table. To resolve this issue, you can discard the rownum field when data records are written to the result table. In most cases, consumers can compute row numbers themselves because the number of top N data records is small. In the preceding example, only the changed records are updated for the result table, rather than nine records. This method helps to significantly reduce the workload for the result table.

Syntax

```
SELECT col1, col2, col3
FROM (
 SELECT col1, col2, col3
  ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
  ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
 FROM table_name)
WHERE rownum <= N [AND conditions]
```

Different from the original Top_N syntax, you need to exclude the rownum field from the second step of the query.

> **Note** If the rownum field is excluded, use caution while defining the primary key for the result table. If the definition is inaccurate, a Top_N query may return inconsistent results. Without the rownum field, the primary key of the result table must be the same as the GROUP BY key that is defined before the Top_N operator.

Example

The following example describes a simplified business scenario for Youku. On Youku, each video generates large amounts of traffic after it is published. The most popular videos can be identified by analyzing the traffic generated by videos. You can return the top 5 videos that generate the most traffic per minute by using the following code.

```
-- Reads data from a Log Service source table.
CREATE TABLE sls_cdnlog_stream (
vid VARCHAR, -- video id
rowtime timestamp, -- Defines the time when the video is played.
```

```
response_size BIGINT, -- Defines the traffic generated by the video.
WATERMARK FOR rowtime as withOffset(rowtime, 0)
) WITH (
type='sls',
...
);
-- Defines a one-minute window to aggregate the bandwidth consumed by the video.
CREATE VIEW cdnvid_group_view AS
SELECT vid,
TUMBLE_START(rowtime, INTERVAL '1' MINUTE) AS start_time,
SUM(response_size) AS rss
FROM sls_cdnlog_stream
GROUP BY vid, TUMBLE(rowtime, INTERVAL '1' MINUTE);
-- Creates the result table.
CREATE TABLE hbase_out_cdnvidtoplog (
vid VARCHAR,
rss BIGINT,
start_time VARCHAR,
  -- Do not define the rownum field in the result table.
  -- Note that the primary key of the result table must be the same as the GROUP BY key that is defined befor
e the Top_N operator.
PRIMARY KEY(start_time, vid)
) WITH (
type='RDS',
...
);
-- Computes and exports the top 5 videos that generate the most traffic per minute.
INSERT INTO hbase_out_cdnvidtoplog
-- Excludes the rownum field from the second step of the query.
SELECT vid, rss, start_time FROM
(
SELECT
vid, start_time, rss,
ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY rss DESC) as rownum,
FROM
cdnvid_group_view
)
WHERE rownum <= 5;
```

Input example

vid (VARCHAR)	rowtime (TIMESTAMP)	response_size (BIGINT)
10000	2017-12-18 15:00:10	2000
10000	2017-12-18 15:00:15	4000
10000	2017-12-18 15:00:20	3000
10001	2017-12-18 15:00:20	3000
10002	2017-12-18 15:00:20	4000
10003	2017-12-18 15:00:20	1000
10004	2017-12-18 15:00:30	1000
10005	2017-12-18 15:00:30	5000
10006	2017-12-18 15:00:40	6000
10007	2017-12-18 15:00:50	8000

Output example

start_time (VARCHAR)	vid (VARCHAR)	rss (BIGINT)
2017-12-18 15:00:00	10000	9000
2017-12-18 15:00:00	10007	8000
2017-12-18 15:00:00	10006	6000
2017-12-18 15:00:00	10005	5000
2017-12-18 15:00:00	10002	4000

3.1.7.9. Complex event processing (CEP)

You can use the MATCH_RECOGNIZE clause to identify the events that match the specified pattern from data streams and export matched events by using a specified method.

Syntax

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[MATCH_RECOGNIZE (
[PARTITION BY {partitionItem [, partitionItem]*}]
[ORDER BY {orderItem [, orderItem]*}]
[MEASURES {measureItem AS col [, measureItem AS col]*}]
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MA
TCH WITH TIMEOUT ROWS]
[AFTER MATCH SKIP]
PATTERN (patternVariable[quantifier] [ patternVariable[quantifier]]*) WITHIN intervalExpression
DEFINE {patternVariable AS patternDefinationExpression [, patternVariable AS patternDefinationExpression
]*}
)];
```

Clause	Description
PARTITION BY	Optional. You can use the PARTITION BY clause to divide the rows of an input table into partitions based on one or more columns.
ORDER BY	Optional. You can use the ORDER BY clause to order the rows within a partition based on one or more columns. If you use multiple columns to order the rows, define `EVENT TIME` or `PROCESSING TIME` as the first column.
MEASURES	You can use the MEASURES clause to define a list of columns for an output table. You can create an expression to compute a value for each column that you have defined.
ONE ROW PER MATCH	Specifies that only one summary row is generated for a pattern match.
ONE ROW PER MATCH WITH TIMEOUT ROWS	Specifies that both a pattern match and timeout can trigger output. In the `PATTERN` clause, you can use the `WITHIN` subclause to define a timeout period.
ALL ROWS PER MATCH	Specifies that for a pattern match that spans multiple rows, each input row triggers one output row.

Clause	Description
ALL ROWS PER MATCH WITH TIMEOUT ROWS	Specifies that both a pattern match and timeout can trigger output. In the `PATTERN` clause, you can use the `WITHIN` subclause to define a timeout period.
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]	Optional. The default value is `ONE ROW PER MATCH`.
AFTER MATCH SKIP TO NEXT ROW	Specifies that pattern matching resumes from the next row that follows the first row of the current match.
AFTER MATCH SKIP PAST LAST ROW	Specifies that pattern matching resumes from the next row that follows the last row of the current match.
AFTER MATCH SKIP TO FIRST patternItem	Specifies that pattern matching resumes from the first row that is mapped to the pattern item.
AFTER MATCH SKIP TO LAST patternItem	Specifies that pattern matching resumes from the last row that is mapped to the pattern item.
PATTERN	You can use the PATTERN clause to specify a regular expression that defines the pattern to be matched. The regular expression is enclosed in parentheses `()`. You can specify multiple pattern variables in the clause. **Note** • If you use a space to separate two pattern variables, no rows are allowed between the two rows that are mapped to the pattern variables. • If you use a `->` string to separate two pattern variables, rows are allowed between the two rows that are mapped to the pattern variables.

- Quantifiers

  `Quantifiers` are used to define the number of iterations for the rows that are mapped to a `pattern variable`.

Quantifier	Description
*	Zero or more iterations

Quantifier	Description
+	One or more iterations
?	Zero or one iteration
{n}	n iterations (n > 0)
{n,}	n or more iterations (n ≥ 0)
{n, m}	From n to m iterations (0 ≤ n ≤ m, 0 < m)
{,m}	Between 0 and m iterations (m > 0)

The quantifiers are greedy by default. For example, if you define `A -> B+` in the PATTERN clause and the input is `a b1, b2, b3`, the output is `a b1, a b1 b2, a b1 b2 b3`. To make a quantifier reluctant, you can add a question mark (`?`) after the quantifier. Supported reluctant quantifiers are listed as follows:

- `*?`
- `+?`
- `{n}?`
- `{n,}?`
- `{n, m}?`
- `{,m}?`

In the preceding example, if you use a reluctant quantifier instead, the output is `a b1, a b2, a b1 b2, a b3, a b2 b3, a b1 b2 b3`.

Note

- You can use the WITHIN clause to define the maximum time span of the rows that match the pattern.
- Syntax of a static window: `INTERVAL 'string' timeUnit [ TO timeUnit ]`. Example: `INTERVAL '10' SECOND, INTERVAL '45' DAY, INTERVAL '10:20' MINUTE TO SECOND, INTERVAL '10:20.10' MINUTE TO SECOND, INTERVAL '10:20' HOUR TO MINUTE, INTERVAL '1-5' YEAR TO MONTH`.
- Syntax of a dynamic window: `INTERVAL intervalExpression`. Example: `INTERVAL A.windowTime + 10`, in which A represents the first pattern variable defined in the PATTERN clause. In an interval expression, you can use the pattern variables defined in the PATTERN clause. Only the first pattern variable can be used. You can use a UDF for an interval expression. The result of the expression indicates the window size (time span), measured in milliseconds. The data type of the result must be LONG.
- You can use the DEFINE clause to specify the definition of a pattern variable. If a pattern variable is not defined in the DEFINE clause, any row can be mapped to the pattern variable.

- Functions in the MEASURES and DEFINE clauses

Function	Description
Row pattern column references	Syntax: `patternVariable.col`. You can use a row pattern column reference to reference the column in all rows that match a `pattern variable`.
PREV	You can only use the PREV function in the DEFINE clause. In most cases, the function is used with `row pattern column references`. You can use the PREV function to reference the value of a column in a previous row based on an offset from the current row. Example: `DOWN AS DOWN.price < PREV(DOWN.price)`. In this example, `PREV(DOWN.price)` indicates the value of the `price` column in the previous row. Note that `DOWN.price` is equivalent to `PREV(DOWN.price, 0)`. `PREV(DOWN.price)` is equivalent to `PREV(DOWN.price, 1)`.
FIRST and LAST	In most cases, the FIRST and LAST functions are used with `row pattern column references`. You can use these functions to reference the value of a column in a row based on an offset from the first or last row that matches a `pattern variable`. For example, `FIRST(A.price, 3)` indicates the value of the price column in the fourth row that matches `Pattern Variable A`. `LAST(A.price, 3)` indicates the value of the price column in the fourth row from the last row that matches `Pattern Variable A`.

- Output methods

Output method	Description
ONE ROW PER MATCH	Output columns include the columns specified in the `PARTITION BY` and `MEASURES` clauses. If a column is specified in the `PARTITION BY` clause, you do not need to define the column again in the `MEASURES` clause.
ONE ROW PER MATCH WITH TIMEOUT ROWS	A pattern match and timeout can both trigger output. In the `PATTERN` clause, you can use the `WITHIN` subclause to define a timeout period.

> Note
>
> - When you define a pattern, you need to define a timeout period for rows that match the pattern. Otherwise, the state may be overwhelming.
> - In the ORDER BY clause, you must define EVENT TIME or PROCESSING TIME as the first column.

Example

- Syntax

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
PARTITION BY symbol
ORDER BY tstamp
MEASURES STRT.tstamp AS start_tstamp,
LAST(DOWN.tstamp) AS bottom_tstamp,
LAST(UP.tstamp) AS end_tstamp
ONE ROW PER MATCH
AFTER MATCH SKIP TO NEXT ROW
PATTERN (STRT DOWN+ UP+) WITHIN INTERVAL '10' SECOND
DEFINE
DOWN AS DOWN.price < PREV(DOWN.price),
UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.start_tstamp;
```

- Input example

timestamp (TIMESTAMP)	card_id (VARCHAR)	location (VARCHAR)	action (VARCHAR)
2018-04-13 12:00:00	1	WW	Tom
2018-04-13 12:05:00	1	WW1	Tom
2018-04-13 12:10:00	1	WW2	Tom
2018-04-13 12:20:00	1	WW	Tom

- Sample code

```
CREATE TABLE datahub_stream (
  `timestamp`          TIMESTAMP,
  card_id              VARCHAR,
```

```
  location            VARCHAR,
  `action`            VARCHAR,
  WATERMARK wf FOR `timestamp` AS withOffset(`timestamp`, 1000)
) WITH (
  type = 'datahub'
  ...
);
CREATE TABLE rds_out (
  start_timestamp          TIMESTAMP,
  end_timestamp            TIMESTAMP,
  card_id                  VARCHAR,
  event                    VARCHAR
) WITH (
  type= 'rds'
  ...
);
-- Example
-- When a credit card specified with a unique card_id is found to have two payment records in different loc
ations within 10 minutes, an alert is triggered. This helps to monitor the unauthorized use of credit cards.
-- Defines the computational logic as follows:
insert into rds_out
select
`start_timestamp`,
`end_timestamp`,
card_id, `event`
from datahub_stream
MATCH_RECOGNIZE (
  PARTITION BY card_id  -- Partitions rows by the card_id column. The rows with the same card ID are allo
cated to the same compute node.
  ORDER BY `timestamp`  -- Orders the rows in a partition based on time.
  MEASURES          -- Defines a list of columns for an output table. You can create an expression to compu
te values for each column that you have defined.
    e2.`action` as `event`,
    e1.`timestamp` as `start_timestamp`,  -- Defines the time of e1 as start_timestamp.
    LAST(e2.`timestamp`) as `end_timestamp`-- Defines the time of e2 as end_timestamp.
  ONE ROW PER MATCH       -- Each match generates an output row.
  AFTER MATCH SKIP TO NEXT ROW-- Resumes the pattern matching from the row that follows the first ro
w of the current match.
  PATTERN (e1 e2+) WITHIN INTERVAL '10' MINUTE  -- Defines two pattern variables: e1 and e2.
  DEFINE              -- Specifies the conditions for the pattern variables defined in the PATTERN clause.
    e1.action = 'Tom',      -- Defines the action of e1 as Tom.
```

```
    e1 as e1.action = 'Tom',  -- Defines the action of e1 as Tom.
    e2 as e2.action = 'Tom' and e2.location <> e1.location -- Defines the action of e2 as Tom. The locations
of e1 and e2 are different.
);
```

- Output example

start_timestamp (TIMESTAMP)	end_timestamp (TIMESTAMP)	card_id (VARCHAR)	event (VARCHAR)
2018-04-13 20:00:00.0	2018-04-13 20:05:00.0	1	Tom
2018-04-13 20:05:00.0	2018-04-13 20:10:00.0	1	Tom

3.1.8. Create a view

In Realtime Compute, you can define a view to query data in one or more tables in a database. You can simplify the development process by using views.

A view displays a logical table that helps to describe the computational logic. It does not physically store any data. To simplify queries, you can use the SQL statement to define a manageable view. A view can be created based on fields of one or more tables in a database. Views do not require any disk space.

Syntax

```
CREATE VIEW viewName
    [ (columnName[ , columnName]*) ]
      AS queryStatement;
```

Example 1

```
CREATE VIEW  LargeOrders(r, t, c, u) AS
 SELECT rowtime,
  productId,
c,
  units
 FROM Orders;
 INSERT INTO rds_output
 SELECT
 r,t,c,u
 FROM LargeOrders;
```

Example 2

Input example

a (VARCHAR)	b (BIGINT)	c (TIMESTAMP)
test1	1	1506823820000
test2	1	1506823850000
test1	1	1506823810000
test2	1	1506823840000
test2	1	1506823870000
test1	1	1506823830000
test2	1	1506823860000

Sample code

```
CREATE TABLE datahub_stream (
a VARCHAR,
b BIGINT,
c TIMESTAMP,
d AS PROCTIME()
) WITH (
 type='datahub',
 endPoint='http://dh-cn-hangzhou.aliyuncs.com',
 project='blink_test',
 topic='window_count',
 accessId='yourAccessId',
 accessKey='yourAccessSecret'
);
CREATE TABLE rds_output (
a varchar,
b TIMESTAMP,
cnt BIGINT,
PRIMARY KEY(a)
)with(
 type = 'rds',
 url='jdbc:mysql://rm-bp1gz4k20****.mysql.rds.aliyuncs.com:3306/****',
 tableName='yourTableName',
 userName='yourAccessUserName',
 password='yourAccessPassword'
);
CREATE VIEW rds_view AS
SELECT a,
  CAST(HOP_START(d, interval '5' second, interval '30' second) AS TIMESTAMP) AS cc,
  sum(b) AS cnt
FROM datahub_stream
GROUP BY HOP(d, interval '5' second, interval '30' second),a;
INSERT INTO rds_output
SELECT
a,
cc,
cnt
FROM rds_view
WHERE cnt=4
```

Output example

a (VARCHAR)	b (TIMESTAMP)	cnt (BIGINT)
test2	2017-11-06 16:54:10	4

3.1.9. Window functions

3.1.9.1. Overview

You can use Flink SQL to aggregate infinite streaming data without defining any window in SQL statements. You can also define a window to aggregate a set of streaming data records as required. For example, you need to calculate the page view for a webpage in the past one minute. You can define a window in the SQL statement to collect data in the past minute and compute the aggregate result.

In Flink SQL, you can aggregate data by using group windows and over windows. The core difference is that when you use over windows, each input data record triggers an aggregate and has an aggregate result. Therefore, over windows are applied to ranking, calculating a moving average, and other scenarios.

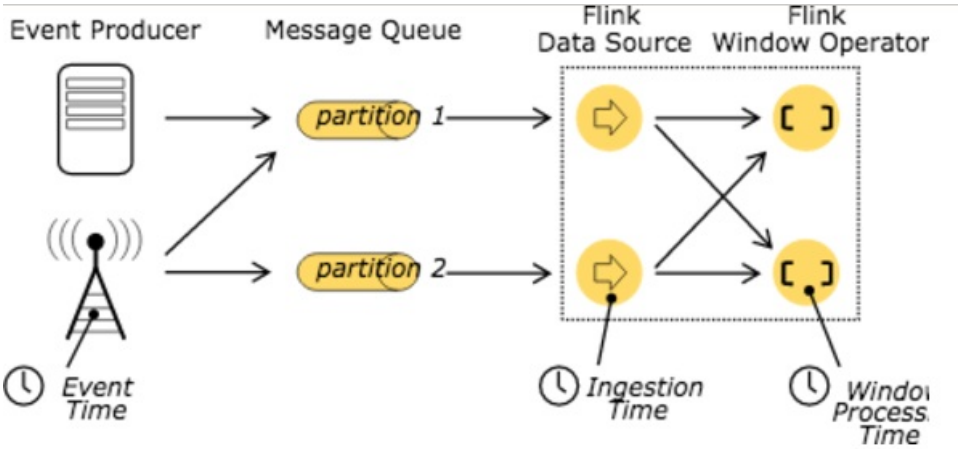
You can aggregate data by using group windows based on two time attributes: event time and processing time. Three types of group windows are available: tumbling window, sliding window, and session window.

Time attributes

Flink SQL supports the following time attributes: event time and processing time.

- Event time is the time when a data record is created. You must define the event time in the table schema.
- Processing time is the local time of the system when the system processes an event.

Positions of different time attributes in the workflow of Realtime Compute



Ingestion time and processing time are the time attributes automatically generated by the system for a streaming data record. Event time is a time attribute that is carried by a steaming data record. Assume that the event time for a streaming data record is t1, and the event time for another streaming data record is t2 (t2 > t1). The first data record may arrive and be processed by Flink later than the second data record because of data disorder, network jitter, or other reasons.

Processing time-based aggregate

Processing time is generated by the system and is not included in source data received from data stores. Therefore, you need to define a processing time in a DDL statement as follows:

```
filedName as PROCTIME()
```

An example of processing time-based aggregate is provided as follows:

```
CREATE TABLE tt_stream (
  a VARCHAR,
  b VARCHAR,
  c BIGINT,
  d AS PROCTIME()
  ) with (
  type = 'sls',
  topic = 'yourTopicName',
  accessId = 'yourAccessId',
  accessKey = 'yourAccessSecret'
  );
  CREATE TABLE rds_output (
  id VARCHAR,
  c TIMESTAMP,
  f TIMESTAMP,
  cnt BIGINT
  ) with (
  type = 'rds',
  url = 'yourDatebaseURL',
  tableName='yourTableName',
  userName='yourDatabaseUserName',
  password='yourDatabasePassword'
  );
  INSERT INTO rds_output
  SELECT a AS id,
  SESSION_START(d, INTERVAL '1' SECOND) AS c,
  SESSION_END(d, INTERVAL '1' SECOND) AS f,
  COUNT(a) AS cnt
  FROM tt_stream
  GROUP BY SESSION(d, INTERVAL '1' SECOND), a
```

Event time-based aggregate

Event time is included in source data received from data stores. You do not need to define an event time in a DDL statement. However, you must use the watermarking mechanism to handle out-of-order data. If you do not define a watermark, data that arrives late may not be processed and the aggregate result is inaccurate.

Watermark

Watermarking is a mechanism in Flink to measure progress in event time. A watermark flows as part of the data stream and carries a timestamp. A watermark is defined in a DDL statement. You can use the following syntax to define a watermark in Flink:

```
WATERMARK [watermarkName] FOR <rowtime_field> AS withOffset(<rowtime_field>, offset)
```

- `watermarkName`: specifies the watermark name. This parameter is optional.
- `<rowtime_field>`: specifies a column (of the `TIMESTAMP` data type) that is used to generate a watermark. This column is labeled as the event time column. When you perform queries by using window functions, you must include a time attribute column in your table.
- `withOffset`: specifies the strategy of generating a waterwark. The value for a watermark is obtained based on `<rowtime_field> - offset`. The first parameter in withOffset must be `<rowtime_field>`.
- `offset`: specifies the offset between a watermark value and an event time. The unit is measured in milliseconds.

For example, a table has a rowtime column of the TIMESTAMP data type, and one value is 1501750584000 (2017-08-03 08:56:24.000). Based on the preceding rowtime column, you can define a watermark with a four-second offset as follows:

```
WATERMARK FOR rowtime AS withOffset(rowtime, 4000)
```

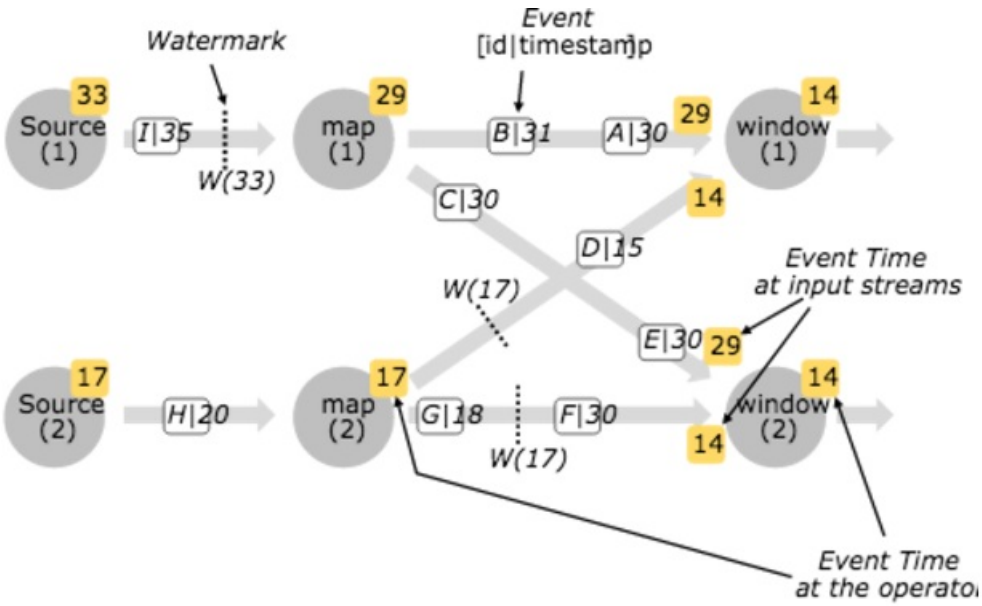
The watermark value of the specified data record is obtained based on the following expression: 1501750584000 - 4000 = 1501750580000 (2017-08-03 08:56:20.000). The watermark indicates that all data with the timestamps earlier than 1501750580000 (2017-08-03 08:56:20.000) has arrived.

> **Note**
>
> - When you define a watermark based on event time, the data type of the rowtime column must be TIMESTAMP. Currently, Realtime Compute only supports 13-bit Unix timestamps measured in milliseconds. If the data type of a rowtime column is not TIMESTAMP or a Unix timestamp is not 13 digits in length, we recommend that you use a computed column expression to convert data types.
> - You can only define event time and processing time in a DDL statement that is used to create a source table.

Summary:

- A watermark indicates that all the events with the timestamps (t') earlier than the watermark (t) have occurred. After a watermark takes effect, all subsequent records with timestamps earlier than the watermark are discarded. This is the current watermarking mechanism in Realtime Compute. However, in the future, improvements to Realtime Compute will allow you to update subsequent data.
- The watermarking mechanism is crucial for handling out-of-order data streams. With watermarks, real-time computing results are accurate even if data streams arrive out of order.
- When an operator has multiple input data streams, the event time of the operator equals the minimum event time of the data streams.

  Diagram of watermarks in parallel data streams



Example of event time-based Aggregation

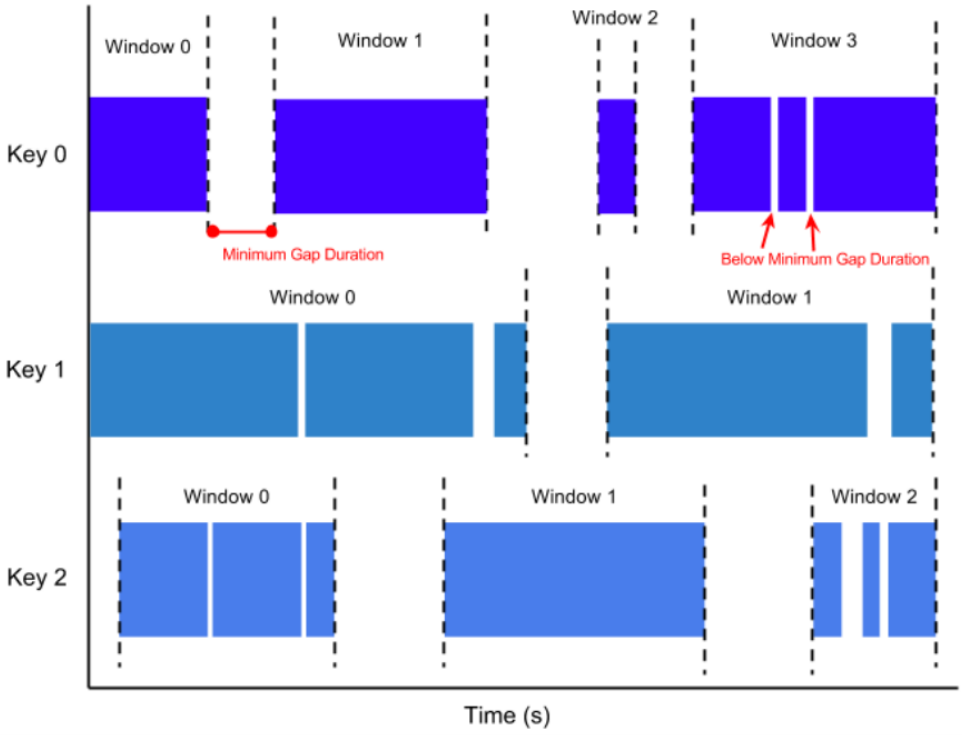
```
CREATE TABLE tt_stream(
  a VARCHAR,
  b VARCHAR,
  c TIMESTAMP,
  WATERMARK wk1 FOR c as withOffset(c, 1000)
  ) with (
  type = 'sls',
  topic = 'blink_tt2tt_test',
  accessId = 'yourAccessId',
  accessKey = 'yourAccessSecret'
  );
  CREATE TABLE rds_output (
  id VARCHAR,
  c TIMESTAMP,
  f TIMESTAMP,
  cnt BIGINT
  ) with (
  type = 'rds',
  url = 'yourDatebaseURL',
  tableName='yourDatabaseTableName',
  userName='yourDatabaseUserName',
  password='yourDatabasePassword'
  );
  INSERT INTO rds_output
  SELECT a AS id,
  SESSION_START(c, INTERVAL '1' SECOND) AS c,
  CAST(SESSION_END(c, INTERVAL '1' SECOND) AS TIMESTAMP) AS f,
  COUNT(a) AS cnt
  FROM tt_stream
  GROUP BY SESSION(c, INTERVAL '1' SECOND), a
```

3.1.9.2. Session window

A session window groups elements by sessions of activities. Unlike tumbling windows and sliding windows, session windows do not overlap and are not fixed in size. If a session window does not receive any elements within a certain period, the session is disconnected and the window is closed.

A session window is configured by using a gap duration, which defines the required length of an inactive period. For example, a data stream that describes click activities may include highly clustered click events, separated by inactive periods. The data that arrives after a specified gap duration is assigned to a new window. Session window shows a session window. Note that each key has different types of windows due to the irregular distribution of elements.

Session window



Syntax

The SESSION function is used to define a session window in a GROUP BY clause.

```
SESSION(<time-attr>, <gap-interval>)
<gap-interval>: INTERVAL 'string' timeUnit
```

> **Note**
>
> The <time-attr> parameter specifies a time attribute based on which a computation is performed. The time attribute is processing time or event time.

Window identifier functions

A window identifier function specifies the start or end time of a window, or the window time attribute. The returned time attribute is used for window joins.

Window identifier functions

Window identifier function	Return type	Description

Window identifier function	Return type	Description
SESSION_START(time-attr, size-interval)	TIMESTAMP	Returns the start time (the boundary value is included) of the window. For example, if the window is [00:10, 00:15), 00:10 is returned.
SESSION_END(time-attr, size-interval)	TIMESTAMP	Returns the end time (the boundary value is included) of the window. For example, if the window is [00:00, 00:15], 00:15 is returned.
SESSION_ROWTIME(time-attr, size-interval)	TIMESTAMP	Returns the end time (the boundary value is excluded) of the window. For example, if the window is [00:00, 00:15], 00:14:59.999 is returned. The return value is a rowtime attribute, based on which time-based operations such as window joins can be performed.

Example

The following example describes how to compute the number of clicks per user during each active session. The session timeout interval is 30 seconds.

```
CREATE TABLE user_clicks(
 username varchar,
 click_url varchar,
 ts timeStamp,
 WATERMARK wk FOR ts as withOffset(ts, 2000)  -- Defines a watermark for rowtime.
) with (
 type='datahub',
 ...
);
CREATE TABLE tumble_output(
 window_start TIMESTAMP,
 window_end TIMESTAMP,
 username VARCHAR,
 clicks BIGINT
) with (
 type='rds'
);
INSERT INTO tumble_output
SELECT
 SESSION_START(ts, INTERVAL '30' SECOND),
 SESSION_END(ts, INTERVAL '30' SECOND),
 username,
 COUNT(click_url)
FROM window_input
GROUP BY SESSION(ts, INTERVAL '30' SECOND), username
```

Input example

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	http://taobao.com/xxx	2017-10-10 10:00:00.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:10.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:49.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:05.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:58.0
Timo	http://taobao.com/xxx	2017-10-10 10:02:10.0

Output example

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 10:00:00.0	2017-10-10 10:00:40.0	Jark	2
2017-10-10 10:00:49.0	2017-10-10 10:01:19.0	Jark	2
2017-10-10 10:01:58.0	2017-10-10 10:02:28.0	Jark	1
2017-10-10 10:02:10.0	2017-10-10 10:02:40.0	Timo	1

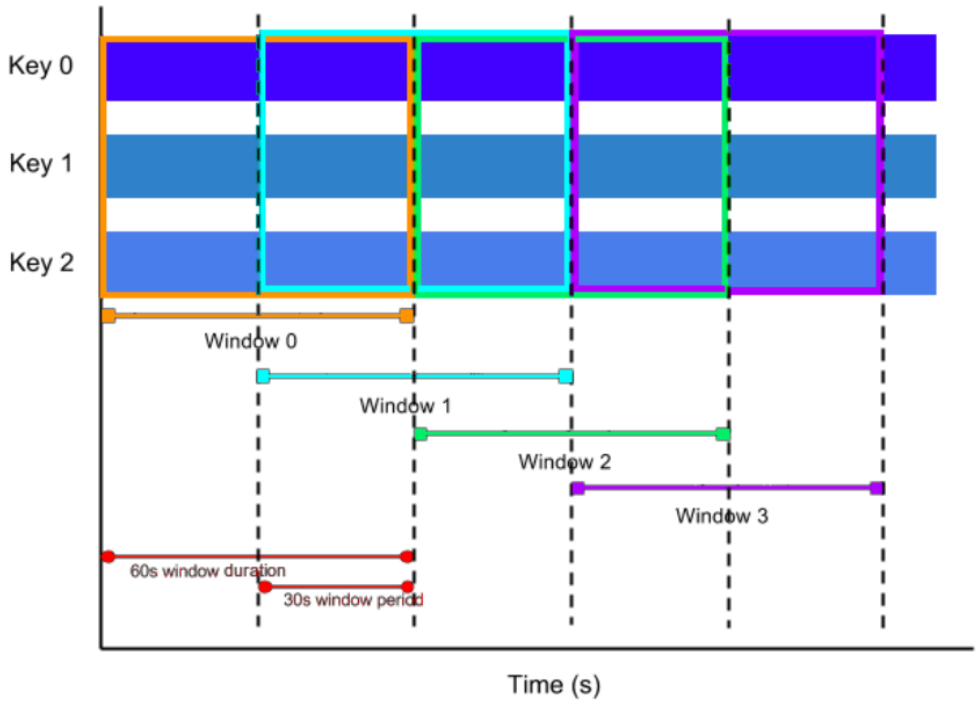
3.1.9.3. Sliding window

Unlike tumbling windows, sliding windows can overlap each other. Sliding windows have two parameters: size and slide. The size parameter specifies the size of a window. The slide parameter defines how frequently a sliding window is started.

- If the slide is smaller than the window size, sliding windows can overlap each other. In this case, elements are assigned to multiple windows.
- If the slide is equal to the size, windows do not overlap and can be considered as tumbling windows.
- If the slide is greater than the size, windows can be considered as tumbling windows that are separated by gaps.

In most cases, multiple windows overlap and most elements are assigned to multiple windows. This type of window is useful in computing moving averages. For example, if the average value of data over the past five minutes needs to be computed every 10 seconds, you can set the size to five minutes and the slide to 10 seconds. The Sliding window figure shows a one-minute window that slides every 30 seconds.

Sliding window



Syntax

The HOP function is used to define a sliding window in a GROUP BY clause.

```
HOP(<time-attr>, <slide-interval>, <size-interval>)
<slide-interval>: INTERVAL 'string' timeUnit
<size-interval>: INTERVAL 'string' timeUnit
```

> **Note**
>
> The <time-attr> parameter specifies a time attribute based on which a computation is performed. The time attribute is processing time or event time.

Window identifier function

A window identifier function specifies the start or end time of a window, or the window time attribute. The returned time attribute is used for window joins.

Window identifier functions

Window identifier function	Return type	Description
HOP_START(time-attr, size-interval)	TIMESTAMP	Returns the start time (the boundary value is included) of the window. For example, if the window is [00:10, 00:15), 00:10 is returned.
HOP_END(time-attr, size-interval)	TIMESTAMP	Returns the end time (the boundary value is included) of the window. For example, if the window is [00:00, 00:15], 00:15 is returned.
HOP_ROWTIME(time-attr, size-interval)	TIMESTAMP	Returns the end time (the boundary value is excluded) of the window. For example, if the window is [00:00, 00:15], 00:14:59.999 is returned. The return value is a rowtime attribute, based on which time-based operations such as window joins can be performed.

Example

The following example describes how to compute the number of clicks per user over the past minute every 30 seconds. That is, a 1-minute window slides every 30 seconds.

```
CREATE TABLE user_clicks(
 username VARCHAR,
 click_url VARCHAR,
 ts TIMESTAMP,
 WATERMARK wk FOR ts as withOffset(ts, 2000) -- Defines a watermark for rowtime.
) with (
 type='datahub',
 ...
);
CREATE TABLE tumble_output(
 window_start TIMESTAMP,
 window_end TIMESTAMP,
 username VARCHAR,
 clicks BIGINT
) with (
 type='rds'
);
INSERT INTO tumble_output
SELECT
 HOP_START(ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
 HOP_END(ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
 username,
 COUNT(click_url)
FROM window_input
GROUP BY HOP(ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE), username
```

Input example

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	http://taobao.com/xxx	2017-10-10 10:00:00.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:10.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:49.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:05.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:58.0
Timo	http://taobao.com/xxx	2017-10-10 10:02:10.0

Output example

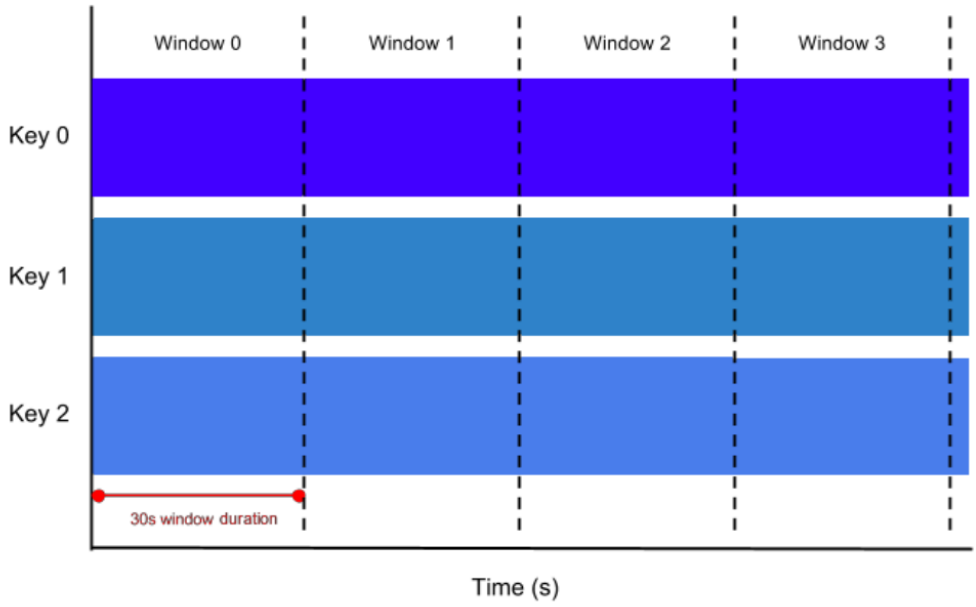
window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	Jark	3
2017-10-10 10:00:30.0	2017-10-10 10:01:30.0	Jark	2
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:01:30.0	2017-10-10 10:02:30.0	Jark	1
2017-10-10 10:01:30.0	2017-10-10 10:02:30.0	Timo	1
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1

3.1.9.4. Tumbling window

In tumbling windows, each element is assigned to a window with a specified size. Tumbling windows are fixed in size and do not overlap.

For example, if a 5-minute tumbling window is defined, elements are assigned into windows based on time periods: [0:00, 0:05), [0:05, 0:10), [0:10, 0:15), and so on. Tumbling window shows a 30-second tumbling window.

Tumbling window



Syntax

The TUMBLE function is used to define a tumbling window in a GROUP BY clause.

```
TUMBLE(<time-attr>, <size-interval>)
<size-interval>: INTERVAL 'string' timeUnit
```

> **Note**
>
> The <time-attr> parameter specifies a time attribute based on which a computation is performed. Time attributes include processing time and event time.

Window identifier function

A window identifier function specifies the start or end time of a window, or the window time attribute, which is used for window joins.

Window identifier functions

Window identifier function	Return type	Description
TUMBLE_START(time-attr, size-interval)	TIMESTAMP	Returns the start time (the boundary value is included) of a window. For example, if the time period of a window is [00:10, 00:15), 00:10 is returned.
TUMBLE_END(time-attr, size-interval)	TIMESTAMP	Returns the end time (the boundary value is included) of a window. For example, if the time period of a window is [00:00, 00:15], 00:15 is returned.
TUMBLE_ROWTIME(time-attr, size-interval)	TIMESTAMP	Returns the end time (the boundary value is excluded) of a window. For example, if the time period of a window is [00:00, 00:15], 00:14:59.999 is returned. The return value is a rowtime attribute, based on which time-based operations such as window joins can be performed.

Example

You can use the following statements to compute the number of clicks per user every one minute.

```
CREATE TABLE user_clicks(
username varchar,
click_url varchar,
ts timeStamp,
WATERMARK wk FOR ts as withOffset(ts, 2000)  -- Defines the rowtime field in the watermark statement.
) with (
type='datahub',
...
);
CREATE TABLE tumble_output(
window_start TIMESTAMP,
window_end TIMESTAMP,
username VARCHAR,
clicks BIGINT
) with (
type='RDS'
);
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE),
TUMBLE_END(ts, INTERVAL '1' MINUTE),
username,
COUNT(click_url)
FROM window_input
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username
```

Input example

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	http://taobao.com/xxx	2017-10-10 10:00:00.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:10.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:49.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:05.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:58.0
Timo	http://taobao.com/xxx	2017-10-10 10:02:10.0

Output example

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	Jark	3
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1

3.1.9.5. Over window

Over window aggregates are known from the standard SQL OVER clause. Different from a group window, an over window is generated for each element, and includes a set of adjacent elements. One element in a stream is assigned to multiple windows. The element that triggers computing of a window determines the last row of the window.

In a data stream that applies over windows, an over window is generated for each element and an aggregate is computed for each over window. Specifically, Realtime Compute stores only one copy of streaming data, and creates an over window for each element. It computes an aggregate for each over window by using the same data copy. Each time the computing is complete, data that is no longer used is deleted from the data copy.

Syntax

```
SELECT
  agg1(col1) OVER (definition1) AS colName,
  ...
  aggN(colN) OVER (definition1) AS colNameN
FROM Tab1
```

> **Note**
>
> - OVER (definition1) in the statement must be the same.
> - The alias specified by AS can be queried through an outer SQL statement.

Type

In Flink SQL, the over window definition follows standard SQL syntax. Over windows are classified into the following two types based on the type of interval each over window uses:

- Over windows that use a row-count interval: Each row of elements is treated as a new computed row. That is, a new window is generated for each row.
- Over windows that use a time internal: All rows of elements with the same timestamp value are treated as one computed row and are assigned to the same window.

Supported time attributes

Over window type	Processing time	Event time
Over window that uses a row-count interval	Yes	Yes
Over window that uses a time interval	Yes	Yes

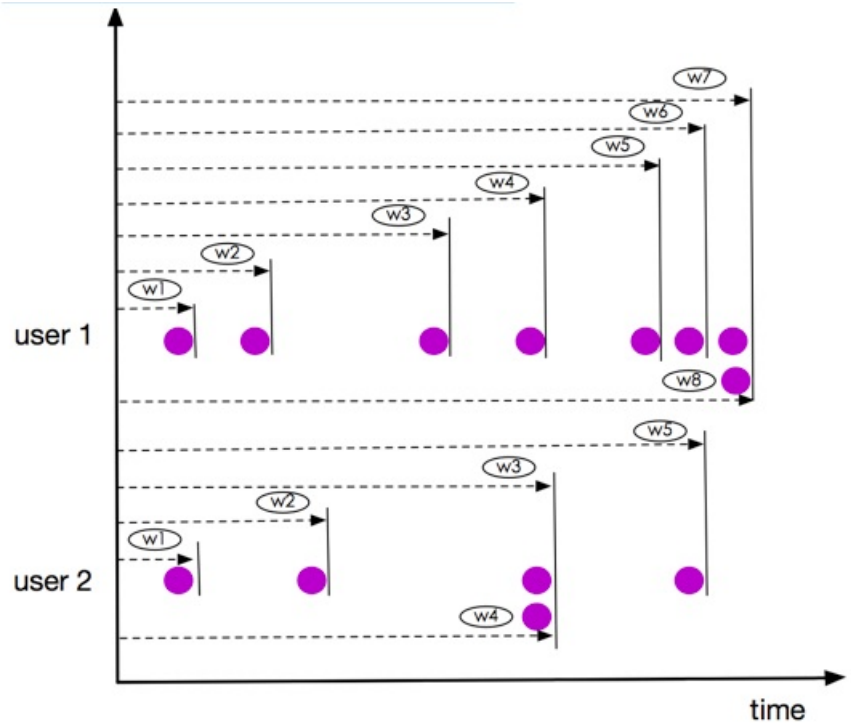
- Over windows that use a row-count interval: The window size is determined by the number of rows of elements.
- Over windows that use a time interval: The window size is determined by a time range.

Over windows that use a row-count interval

- Description

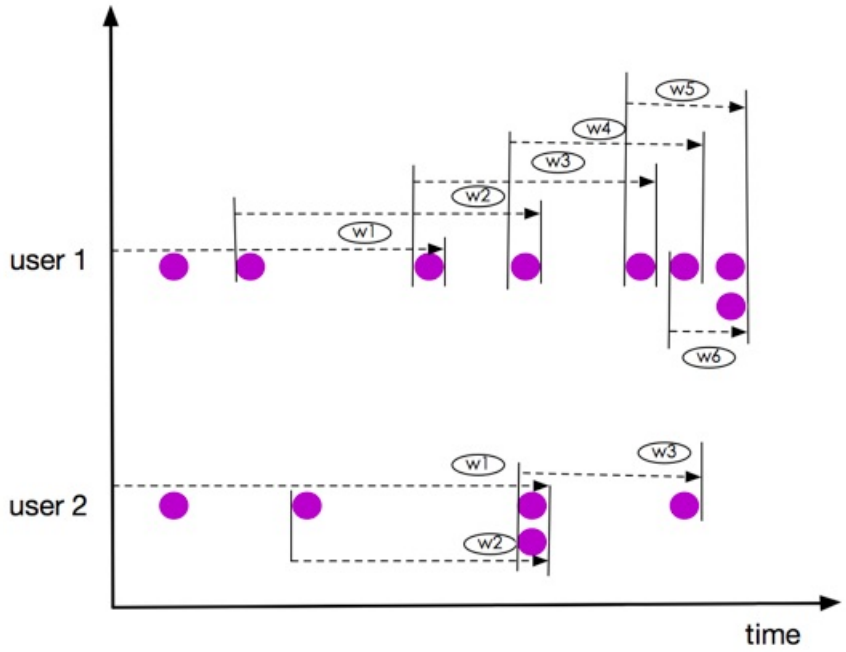
  In over windows that use a row-count interval, each element is assigned to a window and triggers computing. Over windows that use a row-count interval have two types: unbounded and bounded.

  The following figure shows unbounded over windows that use a row-count interval.



> **Note** As shown in the preceding figure, elements in windows w7 and w8 of user 1 arrive at the same time, and so do elements in windows w3 and w4 of user 2. However, the elements are assigned to different windows. In this regard, over windows that use a row-count interval are different from over windows that use a time interval.

The following figure shows bounded over windows that use a row-count interval, in which each window has three elements (first two elements have been assigned to the preceding window).



Note As shown in the preceding figure, elements in windows w5 and w6 of user 1 arrive at the same time, and so do elements in windows w2 and w3 of user 2. However, the elements are assigned to different windows. In this regard, over windows that use a row-count interval are different from over windows that use a time interval.

- Syntax

```
SELECT
  agg1(col1) OVER(
   [PARTITION BY (value_expression1,..., value_expressionN)]
   ORDER BY timeCol
   ROWS
   BETWEEN (UNBOUNDED | rowCount) PRECEDING AND CURRENT ROW) AS colName, ...
FROM Tab1
```

  - value_expression: specifies the value expression used for partitioning.
  - timeCol: specifies the time field used for sorting elements.
  - rowCount: specifies the number of rows that precede the current row and are included in the window.

- Example

  An example of over windows that use a row-count interval is described as follows. Assume a product shelving table, which lists the IDs, types, shelving time, and prices of products. You need to compute the highest price among three similar products before the current product reaches shelves.

  Input example

Item ID	Item type	Launch time	Price
Item 001	Electronic	2017-11-11 10:01:00	20
Item 002	Electronic	2017-11-11 10:02:00	50
Item 003	Electronic	2017-11-11 10:03:00	30
Item 004	Electronic	2017-11-11 10:03:00	60
Item 005	Electronic	2017-11-11 10:05:00	40
Item 006	Electronic	2017-11-11 10:06:00	20
Item 007	Electronic	2017-11-11 10:07:00	70
Item 008	Clothes	2017-11-11 10:08:00	20

Sample code

```
CREATE TABLE tmall_item(
  itemID VARCHAR,
  itemType VARCHAR,
  onSellTime TIMESTAMP,
  price DOUBLE,
  WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
 type = 'sls',
  ...
) ;
SELECT
   itemID,
   itemType,
   onSellTime,
   price,
   MAX(price) OVER (
      PARTITION BY itemType
      ORDER BY onSellTime
      ROWS BETWEEN 2 preceding AND CURRENT ROW) AS maxPrice
 FROM tmall_item
```

Output example

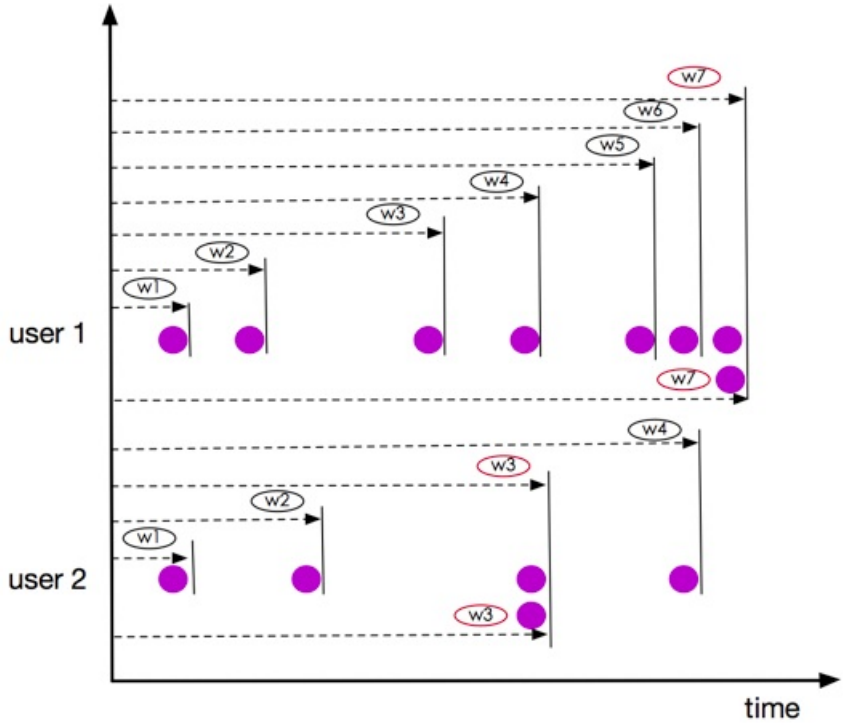
Item ID	Item type	Launch time	Price	Max. Price
Item 001	Electronic	2017-11-11 10:01:00	20	20
Item 002	Electronic	2017-11-11 10:02:00	50	50
Item 003	Electronic	2017-11-11 10:03:00	30	50
Item 004	Electronic	2017-11-11 10:03:00	60	60
Item 005	Electronic	2017-11-11 10:05:00	40	60
Item 006	Electronic	2017-11-11 10:06:00	20	60
Item 007	Electronic	2017-11-11 10:07:00	70	70
Item 008	Clothes	2017-11-11 10:08:00	20	20

Over windows that use a time interval

- Description

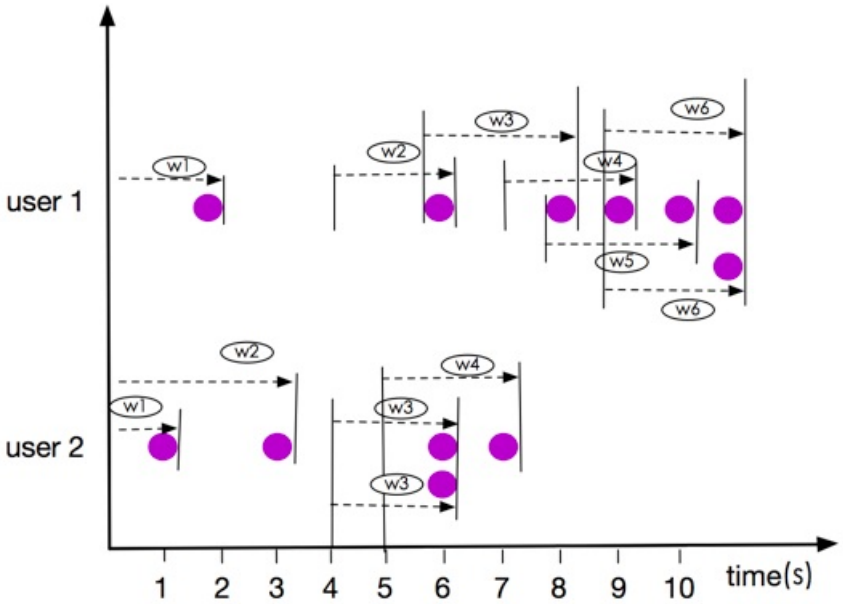
  In over windows that use a time interval, all elements with the same timestamp are assigned to a window and trigger computing. Over windows that use a time interval have two types: unbounded and bounded.

The following figure shows **unbounded over windows that use a time interval**.



Note As shown in the preceding figure, elements in the w7 windows of user 1 arrive at the same time, and so do elements in the w3 windows of user 2. Elements with the same timestamp are assigned to the same window. In this regard, over windows that use a time interval are different from over windows that use a row-count interval.

The following figure shows **bounded over windows that use a time interval**, in which a 3-second window has an interval of 2 seconds .



> **Note** As shown in the preceding figure, elements in the w6 windows of user 1 arrive at the same time, and so do elements in the w3 windows of user 2. Elements with the same timestamps are assigned to the same window. In this regard, over windows that use a time interval are different from over windows that use a row-count interval.

- Syntax

```
SELECT
  agg1(col1) OVER(
   [PARTITION BY (value_expression1,..., value_expressionN)]
   ORDER BY timeCol
   RANGE
   BETWEEN (UNBOUNDED | timeInterval) PRECEDING AND CURRENT ROW) AS colName,
...
FROM Tab1
```

  - value_expression: specifies the value expression used for partitioning.
  - timeCol: specifies the time field used for sorting elements.
  - timeInterval: specifies the time interval between the time of the current row and that of the element row that you trace backwards to.

- Example

  An example of over windows that use a time interval is described as follows. Assume a product shelving table, which lists the IDs, types, shelving time, and prices of products. You need to compute the highest price among similar products that reach shelves 2 minutes earlier than the current product.

  Input example

Item ID	Item type	Launch time	Price
Item 001	Electronic	2017-11-11 10:01:00	20
Item 002	Electronic	2017-11-11 10:02:00	50
Item 003	Electronic	2017-11-11 10:03:00	30
Item 004	Electronic	2017-11-11 10:03:00	60
Item 005	Electronic	2017-11-11 10:05:00	40
Item 006	Electronic	2017-11-11 10:06:00	20
Item 007	Electronic	2017-11-11 10:07:00	70

Item ID	Item type	Launch time	Price
Item 008	Clothes	2017-11-11 10:08:00	20

Sample code

```
CREATE TABLE tmall_item(
  itemID VARCHAR,
  itemType VARCHAR,
  onSellTime TIMESTAMP,
  price DOUBLE,
  WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
 type = 'sls',
  ...
);
SELECT
  itemID,
  itemType,
  onSellTime,
  price,
  MAX(price) OVER (
    PARTITION BY itemType
    ORDER BY onSellTime
    RANGE BETWEEN INTERVAL '2' MINUTE preceding AND CURRENT ROW) AS maxPrice
 FROM tmall_item
```

Output example

Item ID	Item type	Launch time	Price	Max. Price
Item 001	Electronic	2017-11-11 10:01:00	20	20
Item 002	Electronic	2017-11-11 10:02:00	50	50
Item 003	Electronic	2017-11-11 10:03:00	30	50

Item ID	Item type	Launch time	Price	Max. Price
Item 004	Electronic	2017-11-11 10:03:00	60	60
Item 005	Electronic	2017-11-11 10:05:00	40	60
Item 006	Electronic	2017-11-11 10:06:00	20	40
Item 007	Electronic	2017-11-11 10:07:00	70	70
Item 008	Clothes	2017-11-11 10:08:00	20	20

3.1.10. Logical operations

3.1.10.1. =

Syntax

```
A = B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 = int2;
```

- Output example

aa (INT)
97

3.1.10.2. >

Syntax

```
A > B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is greater than B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	65	100

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 > int2;
```

- Output example

aa (INT)
97

3.1.10.3. >=

Syntax

```
A >= B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is greater than or equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65
9	6	61

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 >= int2;
```

- Output example

aa (INT)
97
9

3.1.10.4. <

Syntax

```
A < B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is smaller than B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 < int2;
```

- Output example

aa (INT)
97
9

3.1.10.5. <=

Syntax

```
A <= B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is smaller than or equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 <= int2;
```

- Output example

aa (INT)
97
9

3.1.10.6. <>

Syntax

```
A <> B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is not equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	66	6

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 <> int2;
```

- Output example

aa (INT)
97

3.1.10.7. AND

Syntax

```
A AND B
```

Argument: Both A and B are of the BOOLEAN type.

Description

TRUE is returned if both A and B are TRUE. Otherwise, FALSE is returned.

Sample code

Input example

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1 = 255 AND int3 = 65;
```

Output example

aa (int)
97

3.1.10.8. BETWEEN AND

Syntax

```
A BETWEEN B AND C
```

Arguments: A, B, and C are values of the DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME type.

Description

Use the BETWEEN operator to select a value that is between the two given argument values.

Example 1

Input example

int1 (INT)	int2 (INT)	int3 (INT)
90	80	100
11	10	7

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int1 BETWEEN int2 AND int3;
```

Output example

aa(int)
90

Example 2

Input example

var1 (varchar)	var1 (varchar)	var1 (varchar)
b	a	c

Sample code

```
SELECT var1 as aa
FROM T1
WHERE var1  BETWEEN var2 AND var3;
```

Output example

aa (varchar)
b

Example 3

Input example

TIMESTAMP1 (TIMESTAMP)	TIMESTAMP2 (TIMESTAMP)	TIMESTAMP3 (TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:20	1969-07-20 20:17:45

Sample code

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1  BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

Output example

aa (TIMESTAMP)
1969-07-20 20:17:30

3.1.10.9. IS NOT FALSE

Syntax

```
A IS NOT FALSE
```

Argument: A of the BOOLEAN type

Description

TRUE is returned if A is true. FALSE is returned if A is false.

Examples

Input example

int1(INT)	int2(INT)
255	97

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS NOT FALSE;
```

Output example

aa(int)
97

3.1.10.10. IS NOT NULL

Syntax

```
A IS NOT NULL
```

Arguments: A of the INT type

Description

TRUE is returned if A is null. Otherwise, FALSE is returned.

Examples

Input example

int1(INT)	int2(VARCHAR)
97	null
9	ww123

> **Note** Here, null indicates empty, rather than a null string.

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NOT NULL;
```

Output example

aa(int)
9

3.1.10.11. IS NOT TRUE

Syntax

```
A IS NOT TRUE
```

Argument: A specifies a condition of the BOOLEAN type.

Description

FALSE is returned if the condition is TRUE. TRUE is returned if the condition is FALSE.

Examples

Input example

int1 (INT)	int2 (INT)
255	97

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1 = 25 IS NOT TRUE;
```

Output example

aa (int)
97

3.1.10.12. IS NOT UNKNOWN

Syntax

```
A IS NOT UNKNOWN
```

Argument: A is of the BOOLEAN type.

Description

A is a logical comparison expression, such as 6 < 8.

The value of A is TRUE or FALSE if a logical comparison between two numeric values is performed successfully. If a numeric value is compared with a non-numeric value, the logical comparison fails, and the value of A is neither TRUE nor FALSE. **IS NOT UNKNOWN** is used to check for such failures. It returns **FALSE** if the value of A is neither TRUE nor FALSE and returns **TRUE** if the value of A is TRUE or FALSE.

Examples

Input example

int1 (INT)	int2 (INT)
255	97

Sample code 1

```
SELECT int2 as aa
FROM T1
WHERE int1 = 25 IS NOT UNKNOWN;
```

Output example 1

aa (int)
97

Sample code 2

```
SELECT int2 as aa
FROM T1
WHERE int1 < null IS NOT UNKNOWN;
```

Output example 2

aa (int)
Null

3.1.10.13. IS NULL

Syntax

```
value IS NULL
```

Argument: value specifies a value of any type.

Description

TRUE is returned if A is null. Otherwise, FALSE is returned.

Examples

Input example

int1 (INT)	int2 (VARCHAR)
97	null
9	www

Note In this case, null indicates empty, rather than a null string.

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NULL;
```

Output example

aa (int)
97

3.1.10.14. IS TRUE

Syntax

```
A IS TRUE
```

Argument: A of the BOOLEAN type

Description

TRUE is returned if A is TRUE. FALSE is returned if A is FALSE.

Examples

Input example

int1(INT)	int2(INT)
255	97

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS TRUE;
```

Output example

aa(int)
97

3.1.10.15. IS UNKNOWN

Syntax

```
A IS UNKNOWN
```

Argument: A of the BOOLEAN type

Description

A is a logical comparison expression, such as 6 <> 8.

The value of A is TRUE or FALSE if a logical comparison between two numeric values is performed successfully. If a numeric value is compared with a non-numeric value, the logical comparison fails, and the value of A is neither TRUE nor FALSE. **IS UNKNOWN** is used to check for such failures. It returns **TRUE** if the value of A is neither TRUE nor FALSE and returns **FALSE** if the value of A is TRUE or FALSE.

Examples

Input example

int1(INT)	int2(INT)
255	97

Sample code 1

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS UNKNOWN;
```

Output example 1

aa(int)
Null

Sample code 2

```
SELECT int2 as aa
FROM T1
WHERE int1 > null IS UNKNOWN;
```

Output example 2

aa(int)
97

3.1.10.16. LIKE

Syntax

```
A LIKE B
```

Arguments: A and B of the VARCHAR type

Description

TRUE is returned if A matches B. Otherwise, FALSE is returned.

> **Note** The percent sign (%) can be used as a wildcard.

Examples

Sample code

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

Sample code 1

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR2 LIKE 'ss%';
```

Output example 1

aa(int)
90
9

Sample code 2

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ss';
```

Output example 2

aa(int)
90

Sample code 3

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ho%';
```

Output example 3

aa(int)
99

3.1.10.17. NOT

Syntax

```
NOT A
```

Argument: A of the BOOLEAN type

Description

FALSE is returned if A is TRUE. TRUE is returned if A is FALSE.

Examples

Input example

int2(INT)	int3(INT)
97	65

Sample code

```
SELECT int2 as aa
FROM T1
WHERE  NOT int3=62;
```

Output example

aa(int)
97

3.1.10.18. NOT BETWEEN AND

Syntax

```
A NOT BETWEEN B AND C
```

Arguments: A, B, and C can be of the DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME types.

Description

The NOT BETWEEN operator is used to select a value that is not between the two given argument values.

Example 1

Input example

int1 (INT)	int2 (INT)	int3 (INT)
90	97	80
11	10	7

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int1 NOT BETWEEN int2 AND int3;
```

Output example

aa (int)
11

Example 2

Input example

var1 (varchar)	var1 (varchar)	var1 (varchar)
d	a	c

Sample code

```
SELECT var1 as aa
FROM T1
WHERE var1 NOT BETWEEN var2 AND var3;
```

Output example

aa (varchar)
d

Example 3

Input example

TIMESTAMP1 (TIMESTAMP)	TIMESTAMP2 (TIMESTAMP)	TIMESTAMP3 (TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:40	1969-07-20 20:17:45

Sample code

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 NOT BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

Output example

aa (TIMESTAMP)
1969-07-20 20:17:30

3.1.10.19. OR

Syntax

```
A OR B
```

Arguments: Both A and B are of the BOOLEAN type

Description

TRUE is returned if both A and B are TRUE. Otherwise, FALSE is returned.

Examples

Input example

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1 = 255 OR int3 = 65;
```

Output example

aa (int)
97

3.1.10.20. IN

Syntax

```
SELECT column_name(s)
 FROM table_name
WHERE column_name IN (value1,value2,...)
```

Arguments: Both value1 and value2 must be constants.

Description

Query records that match the arguments.

Examples

Input example

id (INT)	LastName (Varchar)
1	Adams
2	Bush
3	Carter

Sample code

```
SELECT * FROM T1
 WHERE LastName IN ('Adams','Carter')
```

Output example

id (INT)	LastName (Varchar)
1	Adams
3	Carter

3.1.10.21. NOT IN

Syntax

```
SELECT column_name(s)
 FROM table_name
WHERE column_name NOT IN (value1,value2,...)
```

Input parameters: value1 and value2, which must be constants

Description

Searches for records that do not match the values in the arguments.

Example

Input example

id (INT)	LastName (VARCHAR)
1	Adams
2	Bush
3	Carter

Sample code

```
SELECT * FROM T1
 WHERE LastName NOT IN ('Adams','Carter')
```

Output example

id (INT)	LastName (VARCHAR)
2	Bush

3.1.10.22. IS DISTINCT FROM

Syntax

```
A IS DISTINCT FROM B
```

Arguments: A and B of generic types

Description

Check whether two arguments are equal. TRUE is returned if their data types or values are different. FALSE is returned if their data types and values are the same. If either or both arguments are null, they are considered to be the same.

Examples

Input example

A(Int)	B(Varchar)
97	97
null	sss
null	null

Sample code

```
SELECT
A IS DISTINCT FROM B as `result`
FROM T1
```

Output example

result(BOOLEAN)
true
true
false

3.1.10.23. IS NOT DISTINCT FROM

Syntax

```
A IS NOT DISTINCT FROM B
```

Arguments: A and B of generic types

Description

Check whether two arguments are equal. FALSE is returned if their data types or values are different. TRUE is returned if their data types and values are the same. If either or both arguments are null, they are considered to be the same.

Examples

Input example

A(Int)	B(Varchar)
97	97
null	sss
null	null

Sample code

```
SELECT
 A IS NOT DISTINCT FROM B as `result`
FROM T1
```

Output example

result(BOOLEAN)
false
false
true

3.1.11. Built-in functions

3.1.11.1. String functions

3.1.11.1.1. Overview

CHAR_LENGTH

- Syntax: `CHAR_LENGTH(A)`
- Input parameter: The A parameter is of the VARCHAR data type.
- Description: Returns the number of characters in a string.

Example

Input example

var1 (VARCHAR)
ss
231ee

Sample code

```
SELECT CHAR_LENGTH(var1) as aa
FROM T1;
```

Output example

aa (INT)
2
5

ASCII2STR

- Syntax: `string ascii2str(int ascii)`
- Description: Converts a number into an ASCII character. If any argument is NULL, the return value is NULL.
- Input parameter: The value of the ascii parameter ranges from 1 to 127. If the value fall out of the range, an exception occurs.
- Example:

```
ascii2str(9) = '\t'
ascii2str(65) = 'A'
```

CONCAT

- Syntax: `string concat(string a, string b, ...)`
- Description: Concatenates all input strings and returns a new string. If any argument is NULL, the return value is NULL.
- Example:

```
concat('abc', 'def', 'gh') = 'abcdefgh'
concat('abc', NULL) = NULL
```

CONCAT_WS

- Syntax: `string concat_ws(string separator, string a, string b, ...)`
- Description: Concatenates all input strings. Different from the CONCAT function, this function adds a separator between two concatenated strings. If any argument is NULL, the return value is NULL.
- Example:

```
concat_ws(':', 'abc', 'def', 'gh') = 'abc:def:gh'
concat(':', 'abc', NULL) = NULL
```

SUBSTR

- Syntax

```
string substr(string a, int start)
string substr(string a, int start, int len)
```

- Description: Extracts a substring of the specified length from a string, starting from the specified position. If the length is not specified, a substring is extracted from the specified position to the end of the string. The value of the start parameter begins from 1. If the value is negative, a substring is extracted from the end of the string. If any argument is NULL, the return value is NULL.
- Example:

```
substr('galaxy_sql', 5) = 'xy_sql'
substr('galaxy_sql', -5) = 'y_sql'
substr('galaxy_sql', 5, 2) = 'xy'
```

LOWER

- Syntax

```
string lower(string a)
string lcase(string a)
```

- Description: Converts uppercase letters in a string into lowercase letters. If any argument is NULL, the return value is NULL.
- Example: `lower('fOoBaR') = 'foobar'`

UPPER

- Syntax

```
string upper(string a)
string ucase(string a)
```

- Description: Converts lowercase letters in a string into uppercase letters. If any argument is NULL, the return value is NULL.
- Example: `upper('fOoBaR') = 'FOOBAR'`

TRIM

- Syntax: `string trim(string a)`
- Description: Removes the leading and trailing spaces of a string. If any argument is NULL, the return value is NULL.
- Example: `trim(' foobar\t ') = 'foobar\t'`

LTRIM

- Syntax: `string ltrim(string a)`
- Description: Removes the leading spaces of a string. If any argument is NULL, the return value is NULL.
- Example: `ltrim(' foobar ') = 'foobar '`

RTRIM

- Syntax: `string rtrim(string a)`
- Description: Removes the trailing spaces of a string. If any argument is NULL, the return value is NULL.
- Example: `rtrim(' foobar ') = ' foobar'`

LPAD

- Syntax: `string lpad(string str, int len, string pad)`
- Description: Left-pads a string (str) with another string (pad), to the specified length. If any argument is NULL, the return value is NULL.
- Example:

```
lpad('hi', 5, '??') = '??? hi'
lpad('hi', 1, '??') = 'h'
lpad('---', 10, 'abc') = 'abcabca---'
```

RPAD

- Syntax: `string rpad(string str, int len, string pad)`

- Description: Right-pads a string (str) with another string (pad), to the specified length. If any argument is NULL, the return value is NULL.
- Example:

```
rpad('hi', 5, '??') = 'hi???'
rpad('hi', 1, '??') = 'h'
rpad('---', 10, 'abc') = '---abcabca'
```

LENGTH

- Syntax: `int length(string str)`
- Description: Returns the length of a string. If any argument is NULL, the return value is NULL.
- Example:

```
length('hi') = 2
length('I') = 1
length(NULL) = NULL
```

REPEAT

- Syntax: `string repeat(string str, int n)`
- Description: Returns a new string that repeats the specified string for the specified number of times. If any argument is NULL, the return value is NULL.
- Example: `repeat('hi', 2) = 'hihi'`

REVERSE

- Syntax: `string reverse(string str)`
- Description: Reverses a string. If any argument is NULL, the return value is NULL.
- Example: `reverse('abc') = 'cba'`

SPLIT_EX

- Syntax: `string split_ex(string str, string sep, int index)`
- Description: Uses a separator to split a string into several segments and returns the segment identified by the index parameter. If the segment identified by the index parameter does not exist, NULL is returned. The value of the index parameter starts from 0. If any argument is NULL, the return value is NULL.
- Example:

```
split_ex('1.2.3.4', '.', 1) = '2'
split_ex('1.2.3.4', '.', -1) = NULL
split_ex('1.2.3.4', '.', 4) = NULL
```

REGEXP_REPLACE

- Syntax: `string regexp_replace(string str, string pattern, string replacement)`
- Description: Returns a string from the str string with its substrings that match the pattern regular expression replaced with the replacement string.
- Note: Backslashes (\) are used in a regular expression to escape special characters. If any argument is NULL, the return value is NULL.
- Example:

```
regexp_replace('100-200', '(\\d+)', 'num') = 'num-num'
regexp_replace('2014-03-13', '-', '') = '20140313'
regexp_replace('foobar', 'oo|ar', '') = 'fb'
```

REGEXP_EXTRACT

- Syntax: `string regexp_extract(string str, string pattern, int index)`
- Description: Returns a substring of the str string that matches the pattern regular expression. Multiple substrings may all match the pattern. The index parameter identifies the substring to be returned. The value of the index parameter starts from 1.
- Note: A backslash (\) is used in a regular expression to escape special characters. If any argument is NULL, the return value is NULL.
- Example:

```
regexp_extract('foothebar', 'foo(. *?)( bar)', 2) = 'bar'
regexp_extract('100-200', '(\\d+)-(\\d+)', 1) = '100'
```

INSTR

- Syntax: `int instr(string str, string substr)`
- Description: Returns the position where a substring first appears in a string. The return value starts from 1. If the specified substring does not exist in the string, 0 is returned. If any argument is NULL, the return value is NULL.
- Example:

```
instr('foobar', 'foo') = 1
instr('foobar', 'abc') = 0
```

KEYVALUE

- Syntax: `string keyvalue(string str, string split1, string split2, string key_name)`
- Description: Parses key-value pairs from the specified string and returns the value that is mapped to the specified key. If the key does not exist, the return value is NULL. If any argument is NULL, the return value is NULL.
- Example:

```
keyvalue('k1=v1;k2=v2', ';', '=', 'k2') = 'v2'
keyvalue('k1:v1,k2:v2', ',', ':', 'k3') = NULL
```

HASH

- Syntax: `int hash(string str)`
- Description: Returns the absolute value of the integer returned by the hashCode() function that takes the specified string as an input. If any argument is NULL, the return value is NULL.

MD5

- Syntax: `string md5(string str)`
- Description: Returns the MD5 value of a string. If any argument is NULL or an empty string, the return value is an empty string.

3.1.11.1.2. CHAR_LENGTH

Syntax

```
CHAR_LENGTH(A)
```

Input parameter

A: VARCHAR data type

Description

Returns the number of characters in a string.

Example

Input example

var1 (VARCHAR)
ss
231ee

Sample code

```
SELECT  CHAR_LENGTH(var1) as aa
FROM T1;
```

Output example

aa (INT)
2
5

3.1.11.1.3. CHR

Syntax

```
VARCHAR CHR(INT ascii)
```

Arguments

ascii: specifies an integer of the INT type, ranging from 0 to 255. If the argument falls out of this range, null is returned.

Description

Convert ASCII codes into characters.

Examples

Input example

int1(INT)	int2(INT)	int3 (INT)
255	97	65

Sample code

```
SELECT CHR(int1) as var1, CHR(int2) as var2, CHR(int3) as var3
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
ÿ	a	A

3.1.11.1.4. CONCAT

Syntax

```
VARCHAR concat(VARCHAR var1, VARCHAR var2, ...)
```

Arguments

- var1: specifies a common string of the VARCHAR type.
- var2: specifies a common string of the VARCHAR type.

Description

Concatenate two or more strings into a single string. If any argument is NULL, the argument is skipped.

Examples

Input example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
Hello	My	World
Hello	null	World
null	null	World
null	null	null

Sample code

```
SELECT CONCAT(var1, var2, var3) as var
FROM T1
```

Output example

var (VARCHAR)
HelloMyWorld
HelloWorld
World
""

3.1.11.1.5. CONCAT_WS

Syntax

```
VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
```

Arguments

- separator: specifies a separator of the VARCHAR type.
- var1: specifies a value of the VARCHAR type.
- var2: specifies a value of the VARCHAR type.

Description

Concatenate each argument's value with a separator to a new string and return the new string, whose length and type depend on the input values.

> **Note**
> - If the separator value is null, it is treated as an empty string for concatenating the argument's values.
> - If any argument is null, the argument is skipped during concatenation.

Examples

Input example

sep (VARCHAR)	str1 (VARCHAR)	str2 (VARCHAR)	str3 (VARCHAR)
|	Jack	Harry	John
null	Jack	Harry	John
|	null	Harry	John
|	Jack	null	null

Sample code

```
SELECT CONCAT_WS(sep, str1, str2, str3) as var
FROM T1
```

Output example

var (VARCHAR)
Jack|Harry|John

var (VARCHAR)
JackHarryJohn
HarryJJohn
Jack

3.1.11.1.6. FROM_BASE64

Syntax

```
BINARY FROM_BASE64(str)
```

Arguments

str: specifies a Base64 string of the VARCHAR type.

Description

Parse a Base64 string into binary data.

Examples

Input example

a (INT)	b (BIGINT)	c (VARCHAR)
1	1L	null

Sample code

```
SELECT
from_base64(c) as var1,from_base64('SGVsbG8gd29ybGQ=') as var2
FROM T1
```

Output example

var1 (BINARY)	var2 (BINARY)
null	Byte Array: [72('H'), 101('e'), 108('l'), 108('l'), 111('o'), 32(' '), 119('w'), 111('o'), 114('r'), 108('l'), 100('d')]

3.1.11.1.7. HASH_CODE

Syntax

```
INT HASH_CODE(VARCHAR str)
```

Arguments

str: VARCHAR type

Description

Return the absolute value of a string on which hashCode() has been executed.

Examples

Input example

str1(VARCHAR)	str2(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	k1:v1,k2:v2	null

Sample code

```
SELECT  HASH_CODE(str1) as var1, HASH_CODE(str2) as var2, HASH_CODE(nullstr) as var3
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
1099348823	401392878	null

3.1.11.1.8. INITCAP

Syntax

```
INITCAP(A)
```

Input parameter

A: VARCHAR data type

Description

Returns a string with the first letter of each word in uppercase and all other letters in lowercase.

Example

Input example

var1 (VARCHAR)
aADvbn

Sample code

```
SELECT INITCAP(var1) as aa
FROM T1;
```

Output example

aa (VARCHAR)
Aadvbn

3.1.11.1.9. JSON_VALUE

Syntax

```
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
```

Arguments

- content: specifies a JSON object to be parsed, which is represented as a JSON string of the VARCHAR type.
- path: specifies a path expression of the VARCHAR type to parse the JSON objects. The following table lists the path expressions that are currently supported.

Path expressions supported

Symbol	Feature
$	Root object
[]	Array subscript
*	Array wildcard
.	Specified objects in a parent object

Description

Extract the value of the specified path from a JSON string. If the JSON string is invalid or any argument is NULL, the return value is NULL.

Examples

Input example

id (INT)	json(VARCHAR)	path(VARCHAR)
1	[10, 20, [30, 40]]	$[2][*]
2	{"aaa": "bbb", "ccc": {"ddd": "eee", "fff": "ggg", " +""hhh": ["h0", "h1", "h2"]}, "iii": "jjj"}	$.ccc.hhh [*]
3	{"aaa": "bbb", "ccc": {"ddd": "eee", "fff": "ggg"," +""hhh": ["h0", "h1", "h2"]}, "iii": "jjj"}	$.ccc.hhh [1]
4	[10, 20, [30, 40]]	NULL
5	NULL	$[2] [*]
6	“{xx]”	“$[2][*]”

Sample code

```
SELECT
  id,
  JSON_VALUE(json, path) AS value
FROM
  T1
```

Output example

id (INT)	value (VARCHAR)
1	[30, 40]
2	["h0", "h1", "h2"]
3	H1
4	NULL
5	NULL
6	NULL

3.1.11.1.10. KEYVALUE

Syntax

```
VARCHAR KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name)
```

Arguments

- str: specifies a string of key-value pairs. This argument is of the VARCHAR type.
- split1: specifies a delimiter of the VARCHAR type to separate the key-value pairs.
- split2: specifies a delimiter of the VARCHAR type to separate the key and value in a key-value pair.
- key_name: specifies a key value of the VARCHAR type.

Description

Parse the key-value pairs in a string to locate a key-value pair that matches the key-value pair delimiter and key-value delimiter. Then return the key value corresponding to the key name in the key-value pair. If the key name does not exist or an exception occurs, the return value is NULL.

Examples

Input example

str (VARCHAR)	split1 (VARCHAR)	split2 (VARCHAR)	key1 (VARCHAR)
k1=v1; k2=v2	;	=	k2
null	;	|	:
k1:v1|k2:v2	null	=	:
k1:v1|k2:v2	|	=	null
k1:v1|k2:v2	|	=	:
k1:v1|k2:v2	|	=	:

Sample code

```
SELECT  KEYVALUE(str, split1, split2, key1) as result
FROM T1
```

Output example

result (VARCHAR)
v2
null
null
null
null
null

3.1.11.1.11. LOWER

Syntax

```
VARCHAR LOWER(A)
```

Argument

A: specifies a value of the VARCHAR type.

Description

Convert the specified string to a string of lowercase letters.

Examples

Input example

var1 (VARCHAR)
Ss
yyT

Sample code

```
SELECT LOWER(var1) as aa
FROM T1;
```

Output example

aa (VARCHAR)
ss
yyt

3.1.11.1.12. LPAD

Syntax

```
VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
```

Arguments

- str: specifies a source string of the VARCHAR type.
- len: specifies the length of a new string. This argument is of the INT type.

- pad: specifies a string to be repeatedly padded to the source string. This argument is of the VARCHAR type.

Description

Left pad a source string with another string several times until the new string reaches the specified length. If any argument is NULL, the return value is NULL. If the source string is an empty string or the target length is a negative number, the return value is NULL. If the string to be padded to the source string is an empty string, there are two scenarios. A trimmed source string is returned when the specified length is equal to or less than the length of the source string. NULL is returned when the specified length is greater than the length of the source string.

Examples

Input example

str (VARCHAR)	len (INT)	pad (VARCHAR)
""	-2	""
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	""
""	2	s
asd	4	""
""	0	""

Sample code

```
SELECT LPAD(str, len, pad) AS result
FROM T1
```

Output example

result (VARCHAR)
null
JohnJHelloWorld
Jo

result (VARCHAR)
HelC
null
null
as
ss
null
""

3.1.11.1.13. MD5

Syntax

```
VARCHAR MD5(VARCHAR str)
```

Argument

str: specifies a string of the VARCHAR type.

Description

Return the MD5 value of a string. If the argument is NULL, the return value is NULL. If the argument is an empty string, the return value is an empty string.

Examples

Input example

str1 (VARCHAR)	str2 (VARCHAR)
k1 = v1; k2 = v2	null

Sample code

```
SELECT
  MD5(str1) as var1,
  MD5(str2) as var2
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)
19c17f42b4d6a90f7f9ffc2ea9bdd775	NA

3.1.11.1.14. OVERLAY

Syntax

```
VARCHAR OVERLAY ( (VARCHAR x PLACING VARCHAR y FROM INT start_position [ FOR INT length ]) )
```

Arguments

- x: VARCHAR type
- y: VARCHAR type
- start_position: INT type
- length (optional): INT type

Description

Replace a substring of x with y. The replacement starts from start_position and a total of length+1 characters are to be replaced.

Examples

Sample code

```
OVERLAY('abcdefg' PLACING 'hij' FROM 2 FOR 2) as result
FROM  T1
```

Output example

result(VARCHAR)
ahijdefg

3.1.11.1.15. PARSE_URL

Syntax

```
VARCHAR PARSE_URL(VARCHAR urlStr, VARCHAR partToExtract [, VARCHAR key])
```

Arguments

- urlStr: specifies a URL string of the VARCHAR type.
- partToExtract: specifies a value of the VARCHAR type obtained after parsing.
- key: specifies an argument name of the VARCHAR type.

Description

Parse a URL to obtain the value of partToExtract, such as QUERY. The options for partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, FILE, AUTHORITY, and USERINFO.

> **Note** If the partToExtract value is NULL, NULL is returned.

Examples

Input example

url1(VARCHAR)	nullstr(VARCHAR)
http://facebook.com/path/p1.php?query=1	null

Sample code

```
SELECT PARSE_URL(url1, 'QUERY', 'query') as var1,
    PARSE_URL(url1, 'QUERY') as var2,
    PARSE_URL(url1, 'HOST') as var3,
    PARSE_URL(url1, 'PATH') as var4,
    PARSE_URL(url1, 'REF') as var5,
    PARSE_URL(url1, 'PROTOCOL') as var6,
    PARSE_URL(url1, 'FILE') as var7,
    PARSE_URL(url1, 'AUTHORITY') as var8,
    PARSE_URL(nullstr, 'QUERY') as var9,
    PARSE_URL(url1, 'USERINFO') as var10,
    PARSE_URL(nullstr, 'QUERY', 'query') as var11
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)	var11(VARCHAR)
1	query=1	facebook.com	/path/p1.php	null	http	/path/p1.php?query=1	facebook.com	null	null	null

3.1.11.1.16. POSITION

Syntax

```
INTEGER POSITION( x IN y)
```

Arguments

- x: specifies a value of the VARCHAR type.
- y: specifies a value of the VARCHAR type.

Description

Return the position of the first occurrence of string x within string y. The function is similar to LOCATE(substr,str).

Examples

Sample code

```
POSITION('in' IN 'china') as result
FROM  T1
```

Output example

result (INT)
3

3.1.11.1.17. REGEXP

Syntax

```
BOOLEAN REGEXP(VARCHAR str, VARCHAR pattern)
```

Arguments

- str: specifies a string of the VARCHAR type.
- pattern: specifies a pattern of the VARCHAR type.

Description

Perform regular expression matching on a string to check whether it matches the specified pattern. If the string or pattern is empty or null, FALSE is returned.

Examples

Input example

str1(VARCHAR)	pattern1 (VARCHAR)
k1=v1;k2=v2	k2*

str1(VARCHAR)	pattern1 (VARCHAR)
k1:v1|k2:v2	k3
null	k3
k1:v1|k2:v2	null
k1:v1|k2:v2	(

Sample code

```
SELECT  REGEXP(str1, pattern1) as result
FROM T1
```

Output example

result(BOOLEAN)
true
false
null
null
false

3.1.11.1.18. REGEXP_EXTRACT

Syntax

```
VARCHAR REGEXP_EXTRACT(VARCHAR str, VARCHAR pattern, INT index)
```

Arguments

- str: specifies a source string of the VARCHAR type.
- pattern: specifies the regular expression pattern used to match the string to be extracted. This argument is of the VARCHAR type.
- index: specifies the index number of the substring to be extracted from the source string. This argument is of the INT type.

> **Note** Specify constants in the regular expression with Java format because SQL string constants are completely compiled to Java code. For example, you can specify a digit (0-9) in the regular expression as \d, which is the same as the digit in a regular expression in Java code.

Description

Use the regular expression pattern to match and extract the indexed substring from a source string. The index starts from 1. If any argument is NULL or the regular expression is invalid, the return value is NULL.

Examples

Input example

str1 (VARCHAR)	pattern1 (VARCHAR)	index1 (INT)
foothebar	foo(. *?)(bar)	2
100-200	(\d+)-(\d+)	1
null	foo(. *?)(bar)	2
foothebar	null	2
foothebar	""	2
foothebar	(	2

Sample code

```
SELECT REGEXP_EXTRACT(str1, pattern1, index1) as result
FROM T1
```

Output example

result (VARCHAR)
bar
100
null
null
null
null

3.1.11.1.19. REGEXP_REPLACE

Syntax

```
VARCHAR REGEXP_REPLACE(VARCHAR str, VARCHAR pattern, VARCHAR replacement)
```

Arguments

- str: specifies a string of the VARCHAR type.
- pattern: specifies the regular expression pattern used to match the string to be replaced. This argument is of the VARCHAR type.
- replacement: specifies the string used to replace others. This argument is of the VARCHAR type.

> **Note** Specify constants in the regular expression with Java format because SQL string constants are completely compiled to Java code. For example, you can specify a digit (0-9) in the regular expression as \d, which is the same as the digit in a regular expression in Java code.

Description

Replace a substring that matches the regular expression pattern in a source string with a specified string, and return a new string. If any argument is NULL or the regular expression is invalid, the return value is NULL.

Examples

Input example

str1 (VARCHAR)	pattern1 (VARCHAR)	replace1 (VARCHAR)
2014-03-13	-	""
null	-	""
2014-03-13	-	null
2014-03-13	""	s
2014-03-13	(	s
100-200	(\d+)	num

Sample code

```
SELECT REGEXP_REPLACE(str1, pattern1, replace1) as result
FROM T1
```

Output example

result (VARCHAR)
20140313
null
null
2014-03-13
null

result (VARCHAR)

num-num

3.1.11.1.20. REPEAT

Syntax

```
VARCHAR REPEAT(VARCHAR str, INT n)
```

Arguments

- str: specifies the string to be repeated. This argument is of the VARCHAR type.
- n: specifies the number of repetitions. This argument is of the INT type.

Description

Return a new string consisting of N repetitions of the specified string. If any argument is NULL, the return value is NULL. If the value of n is 0 or a negative number, the return value is an empty string.

Examples

Input example

str(VARCHAR)	n(INT)
J	9
Hello	2
Hello	-9
null	9

Sample code

```
SELECT  REPEAT(str,n) as var1
FROM T1
```

Output example

result(VARCHAR)
JJJJJJJJJ
HelloHello

result(VARCHAR)
“”
null

3.1.11.1.21. REVERSE

Syntax

```
VARCHAR REVERSE(VARCHAR str)
```

Arguments

str: specifies a string of the VARCHAR type.

Description

Return a string in the reverse order of the specified string. If the argument is NULL, the return value is NULL.

Examples

Input example

str1(VARCHAR)	str2(VARCHAR)	str3 (VARCHAR)	str4 (VARCHAR)
iPhoneX	Alibaba	World	null

Sample code

```
SELECT  REVERSE(str1) as var1 ,REVERSE(str2) as var2,
    REVERSE(str3) as var3,REVERSE(str4) as var4
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)
XenohPi	ababilA	dlroW	null

3.1.11.1.22. RPAD

Syntax

```
VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
```

Arguments

- str: specifies a source string of the VARCHAR type.
- len: specifies the length of a new string. This argument is of the INT type.
- pad: specifies a string to be repeatedly padded to the source string. This argument is of the VARCHAR type.

Description

Right pad a string with another string several times until the new string reaches the specified length. If any argument is NULL, the return value is NULL. If the source string is an empty string or the specified length is a negative number, the return value is NULL. If the string to be padded to the source string is an empty string, there are two scenarios. A trimmed source string is returned when the specified length is equal to or less than the length of the source string. NULL is returned when the specified length is greater than the length of the source string.

Examples

Input example

str (VARCHAR)	len (INT)	pad (VARCHAR)
""	-2	""
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	""
""	2	s
asd	4	""
""	0	""

Sample code

```
SELECT RPAD(str, len, pad) as result
FROM T1
```

Output example

result (VARCHAR)
null

result (VARCHAR)
HelloWorldJohnJ
Jo
CHel
null
null
as
ss
null
""

3.1.11.1.23. SPLIT_INDEX

Syntax

```
VARCHAR SPLIT_INDEX(VARCHAR str, VARCHAR sep, INT index)
```

Arguments

- str: specifies the string to be split. This argument is of the VARCHAR type.
- sep: specifies the separator of the VARCHAR type.
- index: specifies the index number of the substring to be obtained from the split string. This argument is of the INT type.

Description

Use a separator to split a string into several substrings and return the indexed substring. If no substring is indexed, NULL is returned. The index starts from 0. If any argument is NULL, the return value is NULL.

Examples

Input example

str (VARCHAR)	sep (VARCHAR)	index (INT)
Jack,John,Mary	,	2
Jack,John,Mary	,	3
Jack,John,Mary	null	0
null	,	0

Sample code

```
SELECT SPLIT_INDEX(str, sep, index) as var1
FROM T1
```

Output example

var1 (VARCHAR)
Mary
null
null
null

3.1.11.1.24. STR_TO_MAP

Syntax

```
MAP STR_TO_MAP(VARCHAR text)
MAP STR_TO_MAP(VARCHAR text, VARCHAR listDelimiter, VARCHAR keyValueDelimiter)
```

Arguments

- text: specifies given text of the VARCHAR type.
- listDelimiter: specifies the delimiter used to splits the given text into key-value pairs. This argument is of the VARCHAR type. The default delimiter is a comma (,).
- keyValueDelimiter: specifies the delimiter used to separate the key and value in a key-value pair. This argument is of the VARCHAR type. The default delimiter is an equal sign (=).

> **Note** The delimiters are defined by Java regular expressions. If a special character is used, it needs to be converted to an escape character.

Description

Use listDelimiter to split the given text into key-value pairs, use keyValueDelimiter to separate the key and value in each key-value pair, and then generate and return a map.

Examples

Sample code

```
SELECT
 STR_TO_MAP('k1 = v1, k2 = v2' ) ['k1'] as a
FROM T1
```

Output example

a (VARCHAR)
v1

3.1.11.1.25. SUBSTRING

Syntax

```
VARCHAR SUBSTRING(VARCHAR a, INT start)
VARCHAR SUBSTRING(VARCHAR a, INT start, INT len)
```

Arguments

- a: specifies the string from which a substring is to be obtained. This argument is of the VARCHAR type.
- len: specifies the length of the substring to be obtained. This argument is of the INT type.
- start: specifies the start position where the substring is to be obtained from the specified string. This argument is of the INT type.

Description

Obtain a substring of the specified length from a string, starting from the specified position. If the length is not specified, the entire string is obtained. start begins with 1. If the value is 0, it is regarded as 1. If the value is negative, obtain a substring of the specified length backwards starting from the last character in the string.

Examples

Input example

str (VARCHAR)	nullstr (VARCHAR)
k1 = v1; k2 = v2	null

Sample code

```
SELECT SUBSTRING('', 222222222) as var1,
    SUBSTRING(str, 2) as var2,
    SUBSTRING(str, -2) as var3,
    SUBSTRING(str, -2, 1) as var4,
    SUBSTRING(str, 2, 1) as var5,
    SUBSTRING(str, 22) as var6,
    SUBSTRING(str, -22) as var7,
    SUBSTRING(str, 1) as var8,
    SUBSTRING(str, 0) as var9,
    SUBSTRING(nullstr, 0) as var10
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)	var4 (VARCHAR)	var5 (VARCHAR)	var6 (VARCHAR)	var7 (VARCHAR)	var8 (VARCHAR)	var9 (VARCHAR)	var10 (VARCHAR)
""	1 = v1; k2 = v2	v2	v	1	""	""	k1 = v1; k2 = v2	k1 = v1; k2 = v2	null

3.1.11.1.26. TO_BASE64

Syntax

```
VARCHAR TO_BASE64(bin)
```

Argument

bin: specifies a value of the BINARY type.

Description

Convert the input value of the BINARY type to a Base64 string.

Examples

Input example

c (VARCHAR)
SGVsbG8gd29ybGQ =
SGk =
SGVsbG8 =

Sample code

```
SELECT TO_BASE64(FROM_BASE64(c)) as var1
FROM T1
```

Output example

var1 (VARCHAR)
SGVsbG8gd29ybGQ =
SGk =
SGVsbG8 =

3.1.11.1.27. TRIM

Syntax

```
VARCHAR TRIM( VARCHAR x )
```

Arguments

x: VARCHAR type

Description

Trim leading or trailing characters from a string. The most common use is to trim leading or trailing blank spaces.

Examples

Test example

```
SELECT TRIM('  Sample  '); as result
FROM  T1
```

Output example

result(VARCHAR)
Sample

3.1.11.1.28. UPPER

Syntax

```
VARCHAR UPPER(A)
```

Arguments

A: VARCHAR type

Description

Convert the specified string to a string of uppercase letters.

Examples

Input example

var1(VARCHAR)
ss
ttee

Sample code

```
SELECT UPPER(var1) as aa
FROM T1;
```

Output example

aa(VARCHAR)
SS
TTEE

3.1.11.2. Mathematical unctions

3.1.11.2.1. Addition

Syntax

```
A + B
```

Arguments

A and B: INT type

Description

Return the result of adding B to A.

Examples

Input example

int1(INT)	int2(INT)	int3(INT)
10	20	30

Sample code

```
SELECT int1+int2+int3 as aa
FROM T1;
```

Output example

aa(INT)
60

3.1.11.2.2. Subtraction

Syntax

```
A – B
```

Arguments

A and B: specify values of the INT type.

Description

Return the result of subtracting B from A.

Examples

Input example

int1 (INT)	int2 (INT)	int3 (INT)
10	10	30

Sample code

```
SELECT int3 - int2 - int1 as aa
FROM T1;
```

Output example

aa (INT)
10

3.1.11.2.3. Multiplication

Syntax

```
A * B
```

Arguments

A and B: specify values of the INT type.

Description

Return the result of multiplying A by B.

Examples

Input example

int1 (INT)	int2 (INT)	int3 (INT)
10	20	3

Sample code

```
SELECT int1*int2*int3 as aa
FROM T1;
```

Output example

aa (INT)
600

3.1.11.2.4. Division

Syntax

```
A/B
```

Arguments

A and B: INT type

Description

Return the result of dividing A by B.

Examples

Input example

int1(INT)	int2 (INT)
8	4

Sample code

```
SELECT int1 / int2 as aa
FROM T1;
```

Output example

aa(INT)
2

3.1.11.2.5. ABS

Syntax

```
DOUBLE ABS(A)
```

Arguments

A: DOUBLE type

Description

Return the absolute value of the specified number.

Examples

Input example

int1(DOUBLE)
4.3

Sample code

```
SELECT ABS(in1) as aa
FROM T1;
```

Output example

aa(DOUBLE)
4.3

3.1.11.2.6. ACOS

Syntax

```
DOUBLE ACOS(A)
```

Argument

A: specifies a number of DOUBLE type.

Description

Return the arccos value of the specified number.

Examples

Input example

int1 (DOUBLE)
0.7173560908995228
0.4

Sample code

```
SELECT ACOS(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.7707963267948966
1.1592794807274085

3.1.11.2.7. BIN

Syntax

```
VARCHAR BIN(BIGINT number)
```

Arguments

number: BIGINT type. INT type arguments support implicit conversion, whereas other types of arguments do not.

Description

Convert a value of the BIGINT type to a binary string.

Examples

Input example

id(INT)	x (BIGINT)
1	12L
2	10L
3	0L
4	10000000000L

Sample code

```
SELECT id, bin(x) as var1
FROM T1
```

Output example

id(INT)	var1(VARCHAR)
1	1100
2	1010
3	0
4	1001010100000010111110010000000000

3.1.11.2.8. ASIN

Syntax

```
DOUBLE ASIN(A)
```

Arguments

A: DOUBLE type

Description

Return the arcsine value of the specified number.

Examples

Input example

int1 (DOUBLE)
0.7173560908995228
0.4

Sample code

```
SELECT ASIN(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.8
0.41151684606748806

3.1.11.2.9. ATAN

Syntax

```
DOUBLE ATAN(A)
```

Arguments

A: DOUBLE type

Description

Return the arctangent value of the specified number.

Examples

Input example

int1 (DOUBLE)
0.7173560908995228
0.4

Sample code

```
SSELECT ATAN(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.6222796222326533
0.3805063771123649

3.1.11.2.10. BITAND

Syntax

```
INT BITAND(INT number1,INT number2)
```

Arguments

- number1: specifies a common argument of the INT type.
- number2: specifies a common argument of the INT type.

Description

Perform a bitwise **AND** operation on the specified values. The input and output values are both of the INT type.

Examples

Input example

a(INT)	b(INT)
2	3

Sample code

```
SELECT  BITAND(a, b) as intt
FROM T1
```

Output example

intt(INT)
2

3.1.11.2.11. BITNOT

Syntax

```
INT BITNOT((INT number)
```

Arguments

number: specifies a common argument of the INT type.

Description

Perform a bitwise NOT operation on the specified value. The input and output values are both integers.

Examples

Input example

a(INT)
7

Sample code

```
SELECT  BITNOT(a) as var1,
FROM  T1
```

Output example

var1(INT)
0xfff8

3.1.11.2.12. BITOR

Syntax

```
INT BITOR(INT number1, INT number2)
```

Arguments

- number1: specifies a common argument of the INT type.
- number2: specifies a common argument of the INT type.

Description

Perform a bitwise OR operation on the specified values. The input and output values are both of the INT type.

Examples

Input example

a(INT)	b(INT)
2	3

Sample code

```
SELECT BITOR(a, b) as var1,
FROM T1
```

Output example

var1(INT)
3

3.1.11.2.13. BITXOR

Syntax

```
INT BITXOR(INT number1, INT number2)
```

Arguments

- number1: specifies a common value of the INT type.
- number2: specifies a common value of the INT type.

Description

Return the bitwise XOR (exclusive OR) of two given numbers. The argument and result are both of the INT type.

Examples

Input example

a (INT)	b (INT)
2	3

Sample code

```
SELECT BITXOR(a, b) as var1,
FROM T1
```

Output example

var1 (INT)
1

3.1.11.2.14. CARDINALITY

Syntax

```
CARDINALITY(str)
```

Arguments

str: specifies an array.

Description

Return the number of elements in an array.

Examples

Input example

str(INT)	b (Array[INT])
1	Array(1, 2, 3)

Sample code

```
SELECT cardinality(b) AS result
FROM T1
```

Output example

result(INT)
3

3.1.11.2.15. COS

Syntax

```
DOUBLE COS(A)
```

Arguments

A: DOUBLE type

Description

Return the cosine value of the specified number.

Examples

Input example

in1(DOUBLE)
0.8
0.4

Sample code

```
SELECT COS(in1) as aa
FROM T1;
```

Output example

aa(DOUBLE)
0.6967067093471654
0.9210609940028851

3.1.11.2.16. COT

Syntax

```
DOUBLE COT(A)
```

Arguments

A: specifies a value of the DOUBLE type.

Description

Return the cotangent value of the specified number.

Examples

Input example

in1 (DOUBLE)
0.8

in1 (DOUBLE)
0.4

Sample code

```
SELECT COT(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
1.0296385570503641
0.4227932187381618

3.1.11.2.17. E

Syntax

```
DOUBLE E()
```

Arguments

DOUBLE type

Description

Return the DOUBLE type value of natural constant E.

Examples

Input example

id(INT)	X(DOUBLE)
1	8.0

Sample code

```
SELECT id, e() as dou1, E() as dou2
FROM T1
```

Output example

id(INT)	dou1(DOUBLE)	dou2 (DOUBLE)
1	2.718281828459045	2.718281828459045

3.1.11.2.18. EXP

Syntax

```
DOUBLE EXP(A)
```

Argument

A: specifies a value of the DOUBLE type.

Description

Return the power of e, where e is the base of the natural logarithm.

Examples

Input example

in1 (DOUBLE)
8.0
10.0

Sample code

```
SELECT EXP(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
2980.9579870417283
22026.465794806718

3.1.11.2.19. FLOOR

Syntax

```
DOUBLE FLOOR(A)
```

Arguments

A: DOUBLE type

Description

Return the maximum integer that is less than or equal to the provided number.

Examples

Input example

in1(DOUBLE)
8.123
10.321

Sample code

```
SELECT FLOOR(in1) as aa
FROM T1;
```

Output example

aa(INT)
8
10

3.1.11.2.20. LN

Syntax

```
DOUBLE ln(DOUBLE number)
```

Arguments

number: specifies a number of the DOUBLE type. If the input value is of the VARCHAR or BIGINT type, it is implicitly converted to the DOUBLE type before an operation.

Description

Return the natural logarithm of the specified number. The return value is a logarithm of the DOUBLE type.

Examples

Input example

ID (INT)	X (DOUBLE)
1	100.0
2	8.0

Sample code

```
SELECT id, ln(x) as dou1, ln(e()) as dou2
FROM  T1
```

Output example

ID (INT)	dou1 (DOUBLE)	dou2 (DOUBLE)
1	4.605170185988092	1.0
2	2.0794415416798357	1.0

3.1.11.2.21. LOG

Syntax

```
DOUBLE LOG(DOUBLE base, DOUBLE x)
DOUBLE LOG(DOUBLE x)
```

Arguments

- base: specifies a value of the DOUBLE type.
- x: specifies a value of the DOUBLE type.

Description

Return the natural logarithm of x to the base specified by the base argument. The return value is a logarithm of the DOUBLE type. If there is only one argument value, the return value is a natural logarithm to the base specified by the argument.

Examples

Input example

ID (INT)	BASE (DOUBLE)	X (DOUBLE)
1	10.0	100.0
2	2.0	8.0

Sample code

```
SELECT id, LOG(base, x) as dou1, LOG(2) as dou2
FROM  T1
```

Output example

ID (INT)	dou1 (DOUBLE)	dou2 (DOUBLE)
1	2.0	0.6931471805599453
2	3.0	0.6931471805599453

3.1.11.2.22. LOG10

Syntax

```
DOUBLE LOG10(DOUBLE x)
```

Argument

x: specifies a value of the DOUBLE type.

Description

Return the base-10 logarithm of x. If x is NULL, the return value is NULL. If x is a negative number, an exception occurs.

Examples

Input example

id (INT)	X (INT)
1	100
2	10

Sample code

```
SELECT id, log10(x) as dou1
FROM T1
```

Output example

ID (INT)	dou1 (DOUBLE)
1	2.0
2	1.0

3.1.11.2.23. LOG2

Syntax

```
DOUBLE LOG2(DOUBLE x)
```

Arguments

x: specifies a multiple of 2. This argument is of the DOUBLE type.

Description

Return the base-2 logarithm of x. If x is NULL, the return value is NULL. If x is a negative number, an exception occurs.

Examples

Input example

id (INT)	X (INT)
1	8
2	2

Sample code

```
SELECT id, log2(x) as dou1
FROM T1
```

Output example

ID (INT)	dou1 (DOUBLE)
1	3.0
2	1.0

3.1.11.2.24. PI

Syntax

```
DOUBLE PI( )
```

Description

Obtain the ratio of circumference to diameter, namely, Pi.

Examples

Input example

id (INT)	X (INT)
1	8

Sample code

```
SELECT id, PI() as dou1
FROM T1
```

Output example

ID (INT)	dou1 (DOUBLE)
1	3.141592653589793

3.1.11.2.25. POWER

Syntax

```
DOUBLE POWER(A, B)
```

Input parameters

A and B: DOUBLE data type

Description

Returns the result of raising A to the power of B.

Example

Input example

int1 (DOUBLE)	in2 (DOUBLE)
2.0	4.0

Sample code

```
SELECT POWER(in1, in2) as aa
FROM T1;
```

Output example

aa (DOUBLE)
16.0

3.1.11.2.26. RAND

Syntax

```
DOUBLE RAND([BIGINT seed])
```

Arguments

seed: specifies a random number of the BIGINT type. This argument specifies the start number of a random number sequence.

Description

Return a random number of the DOUBLE type. The return value range is `[0, 1)`.

Examples

Input example

id(INT)	X(INT)
1	8

Sample code

```
SELECT id, rand(1) as dou1, rand(3) as dou2
FROM T1
```

Output example

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	0.7308781907032909	0.731057369148862

3.1.11.2.27. SIN

Syntax

```
DOUBLE SIN(A)
```

Arguments

A: specifies a number of the DOUBLE type.

Description

Return the sine value of the specified number.

Examples

Input example

in1 (DOUBLE)
8.0
0.4

Sample code

```
SELECT SIN(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.9893582466233818
0.3894183423086505

3.1.11.2.28. SQRT

Syntax

```
DOUBLE SQRT(A)
```

Arguments

A: specifies a number of the DOUBLE type.

Description

Return the square root of the specified number.

Examples

Input example

in1 (DOUBLE)
8.0

Sample code

```
SELECT SQRT(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
2.8284271247461903

3.1.11.2.29. TAN

Syntax

```
DOUBLE TAN(A)
```

Arguments

A: DOUBLE type

Description

Return the tangent value of the specified number.

Examples

Input example

in1(DOUBLE)
8.0
0.4

Sample code

```
SELECT TAN(in1) as aa
FROM T1;
```

Output example

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

3.1.11.2.30. CEIL

Syntax

```
value_class CEIL(value)
```

Arguments

value: specifies a value of the INT, DOUBLE, or BIGINT type.

Description

Return the smallest integer greater than or equal to a specified value. The value_class data type returned by CEIL is the same as that of the argument.

Examples

Input example

ID (INT)	value1 (INT)	value2 (DOUBLE)
1	3	3.6

Sample code

```
SELECT CEIL (value2) as result1
SELECT CEIL (value2) as result2
FROM  T1
```

Output example

ID (INT)	result1 (INT)	result2 (DOUBLE)
1	3	4.0

3.1.11.2.31. CHARACTER_LENGTH

Syntax

```
INTEGER CHARACTER_LENGTH( VARCHAR x )
```

Arguments

x: specifies a value of the VARCHAR type.

Description

Return the number of characters contained in a string.

Examples

Input example

ID (INT)	X (VARCHAR)
1	StreamCompute

Sample code

```
SELECT CHARACTER_LENGTH( x ) as result
FROM  T1
```

Output example

ID (INT)	result (INT)
1	13

3.1.11.2.32. DEGREES

Syntax

```
DOUBLE DEGREES(double x)
```

Arguments

x: specifies a radian value of the DOUBLE type.

Description

Return the result of converting radians to degrees.

Examples

Sample code

```
SELECT DEGREES( PI() ) as result
FROM  T1
```

Output example

result(DOUBLE)
180.0

3.1.11.2.33. MOD

Syntax

```
Integer MOD(Integer x,Integer y)
```

Arguments

- x: specifies a value of the Integer type.
- y: specifies a value of the Integer type.

Description

Return the remainder when integer x is divided by integer y, without considering the quotient. When x is a negative integer or both x and y are negative integers, the result is negative.

Examples

Input example

X (INT)	Y (INT)
29	3
-29	3
-29	-3

Sample code

```
SELECT MOD(x,y) as result
FROM  T1
```

Output example

ID (INT)	result (INT)
1	2
2	-2
3	-2

3.1.11.2.34. ROUND

Syntax

```
DECIMAL ROUND(DECIMAL x, INT n)
```

Argument

x: DECIMAL type

Description

Round the x argument's value to n decimal places.

Examples

Input example

in1 (DECIMAL)
0.7173560908995228
0.4

Sample code

```
SELECT ROUND(in1,2) as `result`
FROM T1;
```

Output example

result (DECIMAL)
0.72
0.40

3.1.11.3. Date functions

3.1.11.3.1. CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP
```

Arguments

None

Description

Return the current UTC (GMT+0) timestamp, which is eight hours earlier than Beijing time.

Examples

Sample code

```
SELECT CURRENT_TIMESTAMP as var1
FROM T1
```

Output example

var1 (TIMESTAMP)
2007-04-30 13:10:02.047

3.1.11.3.2. DATEDIFF

Syntax

```
INT DATEDIFF(VARCHAR enddate, VARCHAR startdate)
INT DATEDIFF(TIMESTAMP enddate, VARCHAR startdate)
INT DATEDIFF(VARCHAR enddate, TIMESTAMP startdate)
INT DATEDIFF(TIMESTAMP enddate, TIMESTAMP startdate)
```

Arguments

- startdate: specifies a start date of the TIMESTAMP or VARCHAR type. The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
- enddate: specifies an end date of the TIMESTAMP or VARCHAR type. The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.

Description

Compute the number of days between the specified start date and end date. The return value is an integer of the TIMESTAMP type or in the format of `yy-MM-dd HH:mm:ss` or `yy-MM-dd`. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

datetime1(VARCHAR)	datetime2 (VARCHAR)	nullstr(VARCHAR)
2017-10-15 00:00:00	2017-09-15 00:00:00	null

Sample code

```
SELECT  DATEDIFF(datetime1, datetime2) as int1,
    DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',datetime2) as int2,
    DATEDIFF(datetime2,TIMESTAMP '2017-10-15 23:00:00') as int3,
    DATEDIFF(datetime2,nullstr) as int4,
    DATEDIFF(nullstr,TIMESTAMP '2017-10-15 23:00:00') as int5,
    DATEDIFF(nullstr,datetime2) as int6,
    DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',TIMESTAMP '2017-9-15 00:00:00')as int7
FROM T1
```

Output example

int1(INT)	int2(INT)	int3(INT)	int4(INT)	int5(INT)	int6(INT)	int7 (INT)
30	31	-31	null	null	null	31

3.1.11.3.3. DATE_ADD

Syntax

```
VARCHAR DATE_ADD(VARCHAR startdate, INT days)
VARCHAR DATE_ADD(TIMESTAMP time, INT days)
```

Arguments

- startdate: specifies a date of the VARCHAR type. The format is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
- time: specifies a date of the TIMESTAMP type.
- days: specifies the number of days of the INT type.

Description

Return a date that is X (specified by days) days after the specified date of the VARCHAR type. The date format can be yyyy-MM-dd hh:mm:ss, yyyy-MM-dd, and timestamp string. The return value is a string type date in the format of yyyy-MM-dd. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

datetime1 (VARCHAR)	nullstr (VARCHAR)
2017-09-15 00:00:00	null

Sample code

```
SELECT DATE_ADD(datetime1, 30) as var1,
    DATE_ADD(TIMESTAMP '2017-09-15 23:00:00',30) as var2,
    DATE_ADD(nullstr,30) as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
2017-10-15	2017-10-15	null

3.1.11.3.4. DATE_FORMAT

Syntax

```
VARCHAR DATE_FORMAT(TIMESTAMP time, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR from_format, VARCHAR to_format)
```

Input parameters

- date: specifies a date of the VARCHAR data type. The default format is yyyy-MM-dd HH:mm:ss.
- time: specifies a date of the TIMESTAMP data type.
- from_format: specifies a source date format.
- to_format: specifies a target date format.

Description

Converts a specified date from a source format to a target format. The date or time parameter specifies a date. The from_format parameter is optional and specifies the source date format. The default format is yyyy-MM-dd hh:mm:ss. The to_format parameter specifies the target date format. The return value is a date in the specified target format. If any argument is NULL or a parsing error occurs, the return value is NULL.

Example

Input example

date1 (VARCHAR)	datetime1 (VARCHAR)	nullstr (VARCHAR)
0915-2017	2017-09-15 00:00:00	null

Sample code

```
SELECT  DATE_FORMAT(datetime1, 'yyMMdd') as var1,
    DATE_FORMAT(nullstr, 'yyMMdd') as var2,
    DATE_FORMAT(datetime1, nullstr) as var3,
    DATE_FORMAT(date1, 'MMdd-yyyy', nullstr) as var4,
    DATE_FORMAT(date1, 'MMdd-yyyy', 'yyyyMMdd') as var5,
    DATE_FORMAT(TIMESTAMP '2017-09-15 23:00:00', 'yyMMdd') as var6
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)	var4 (VARCHAR)	var5 (VARCHAR)	var6 (VARCHAR)
170915	null	null	null	20170915	170915

3.1.11.3.5. DATE_SUB

Syntax

```
VARCHAR DATE_SUB(VARCHAR startdate, INT days)
VARCHAR DATE_SUB(TIMESTAMP time, INT days)
```

Arguments

- startdate: specifies a date of the VARCHAR type. The format is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
- time: specifies a date of the TIMESTAMP type.
- days: specifies the number of days. This argument is of the INT type.

Description

Return the date that is X (specified by days) days before the specified date. The return value is a VARCHAR type date in the format of yyyy-MM-dd. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

date1 (VARCHAR)	nullstr (VARCHAR)
2017-10-15	null

Sample code

```
SELECT  DATE_SUB(date1, 30) as var1,
    DATE_SUB(TIMESTAMP '2017-10-15 23:00:00',30) as var2,
    DATE_SUB(nullstr,30) as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
2017-09-15	2017-09-15	null

3.1.11.3.6. DAYOFMONTH

Syntax

```
INT DAYOFMONTH(TIMESTAMP time)
INT DAYOFMONTH(DATE date)
```

Arguments

- date: specifies a date of the DATE type.
- time: specifies a date of the TIMESTAMP type.

Description

Return the day number of the specified timestamp or date. Valid range: [1,31].

Examples

Input example

tsStr (VARCHAR)	dateStr (VARCHAR)	tdate (DATE)	ts (TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT DAYOFMONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
    DAYOFMONTH(DATE '2017-09-22') as int2,
    DAYOFMONTH(tdate) as int3,
    DAYOFMONTH(ts) as int4,
    DAYOFMONTH(CAST(dateStr AS DATE)) as int5,
    DAYOFMONTH(CAST(tsStr AS TIMESTAMP))  as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
15	22	10	15	15	15

3.1.11.3.7. FROM_UNIXTIME

Syntax

```
VARCHAR FROM_UNIXTIME(BIGINT unixtime[, VARCHAR format])
```

Arguments

- unixtime: specifies a Unix timestamp (in seconds) of the BIGINT type. An exception occurs if this argument is of another type.

- format: specifies the target date format of the VARCHAR type. The default format is yyyy-MM-dd hh:mm:ss. This argument is optional.

Description

Return a VARCHAR type date in the specified date format. The default format is yyyy-MM-dd HH:mm:ss. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

unixtime1(INT)	nullstr(VARCHAR)
1505404800	null

Sample code

```
SELECT FROM_UNIXTIME(unixtime1) as var1,
    FROM_UNIXTIME(unixtime1,'MMdd-yyyy') as var2,
    FROM_UNIXTIME(unixtime1,nullstr) as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
2017-09-15 00:00:00	0915-2017	null

3.1.11.3.8. HOUR

Syntax

```
INT HOUR(TIMESTAMP timestamp)
INT HOUR(TIME time)
```

Arguments

- time: specifies a time value of the TIME type.
- time: specifies a time value of the TIMESTAMP type.

Description

Return the hours (in 24-hour format) from the specified time or timestamp string. The return value ranges from 0 to 23.

Examples

Input example

datetime1 (VARCHAR)	time1 (VARCHAR)	time2 (TIME)	timestamp1 (TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

Sample code

```
SELECT HOUR(TIMESTAMP '2016-09-20 23:33:33') as int1,
    HOUR(TIME '23:30:33') as int2,
    HOUR(time2) as int3,
    HOUR(timestamp1) as int4,
    HOUR(CAST(time1 AS TIME)) as int5,
    HOUR(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
23	23	22	11	22	11

3.1.11.3.9. LOCALTIME

Syntax

```
TIME LOCALTIME
```

Arguments

The function is a non-parameterized constructor and can be used directly as a variable.

Description

Return the current time of the TIME type in the session time zone.

Examples

Sample code

```
SELECT  LOCALTIME as `result`
FROM T1
```

Output example

result(TIME)
19:00:47

3.1.11.3.10. MINUTE

Syntax

```
INT MINUTE(TIMESTAMP timestamp)
 INT MINUTE(TIME time)
```

Arguments

- time: specifies a time value of the TIME type.
- time: specifies a time value of the TIMESTAMP type.

Description

Return the minutes from the specified time value. Valid range: [0, 59].

Examples

Input example

datetime1 (VARCHAR)	time1 (VARCHAR)	time2 (TIME)	timestamp1 (TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

Sample code

```
SELECT MINUTE(TIMESTAMP '2016-09-20 23:33:33') as int1,
    MINUTE(TIME '23:30:33') as int2,
    MINUTE(time2) as int3,
    MINUTE(timestamp1) as int4,
    MINUTE(CAST(time1 AS TIME)) as int5,
    MINUTE(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
33	30	23	12	23	12

3.1.11.3.11. MONTH

Syntax

```
INT MONTH(TIMESTAMP time)
 INT MONTH(DATE date)
```

Arguments

- time: specifies a time value of the TIMESTAMP type.
- date: specifies a time value of the TIME type.

Description

Return the month number of the specified time or date. Valid range: [1, 12].

Examples

Input example

tsStr (VARCHAR)	dateStr (VARCHAR)	tdate (DATE)	ts (TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT MONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
    MONTH(DATE '2017-09-22') as int2,
    MONTH(tdate) as int3,
    MONTH(ts) as int4,
    MONTH(CAST(dateStr AS DATE)) as int5,
    MONTH(CAST(tsStr AS TIMESTAMP))  as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
9	9	11	10	9	10

3.1.11.3.12. NOW

Syntax

```
BIGINT NOW()
```

Arguments

If no argument is specified, the Unix timestamp (in seconds) of the current system time is returned.

Description

Return the Unix timestamp (in seconds) of the current time zone. You can also specify an INT type argument as an offset (in seconds) and use UNIX_TIMESTAMP(CAST(NOW AS VARCHAR)) to convert the INT type argument to a timestamp value. Then add the offset to the current timestamp to return a value.

Examples

Input example

b(VARCHAR)
null

Sample code

```
SELECT  NOW() as big1, NOW(b) as big2
FROM T1
```

Output example

big1(BIGINT)	big2 (BIGINT)
1403006911	null

3.1.11.3.13. SECOND

Syntax

```
INT SECOND(TIMESTAMP timestamp)
 INT SECOND(TIME time)
```

Arguments

- time: specifies a time value of the TIME type.
- time: specifies a time value of the TIMESTAMP type.

Description

Extract the seconds from the specified time value. The return value ranges from 0 to 59.

Examples

Input example

datetime1(VARCHAR)	time1 (VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

Sample code

```
SELECT SECOND(TIMESTAMP '2016-09-20 23:33:33') as int1,
    SECOND(TIME '23:30:33') as int2,
    SECOND(time2) as int3,
    SECOND(timestamp1) as int4,
    SECOND(CAST(time1 AS TIME)) as int5,
    SECOND(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

Output example

int1(INT)	int2(INT)	int3(INT)	int4 (INT)	int5 (INT)	int6 (INT)
33	33	24	13	24	13

3.1.11.3.14. TIMESTAMPADD

Syntax

```
TIMESTAMP TIMESTAMPADD(interval,INT int_expr,TIMESTAMP datetime_expr)
DATE TIMESTAMPADD(interval,INT int_expr,DATE datetime_expr)
```

Input parameters

- interval

FRAC_SECOND	Specifies an interval in milliseconds.
SECOND	Specifies an interval in seconds.
MINUTE	Specifies an interval in minutes.
HOUR	Specifies an interval in hours.
DAY	Specifies an interval in days.
WEEK	Specifies an interval in weeks.
MONTH	Specifies an interval in months.
QUARTER	Specifies an interval in quarters.

YEAR	Specifies an interval in years.

- int_expr: specifies an increment that is an integer.
- datetime_expr: specifies a timestamp or date to which the increment is to be added.

Description

The data type of a result returned by the TIMESTAMPADD function is the same as that of datetime_expr. The TIMESTAMPADD function adds an increment (int_expr) to a timestamp or date (datetime_expr) and returns the updated time.

Example

Input example

a (TIMESTAMP)	b (DATE)
2018-07-09 10:23:56	1990-02-20

Sample code

```
SELECT
TIMESTAMPADD(HOUR,3,a) AS `result1`,
TIMESTAMPADD(DAY,3,b) AS `result2`
FROM T1
```

Output example

result1 (TIMESTAMP)	result2 (DATE)
2018-07-09 13:23:56	1990-02-23

3.1.11.3.15. TO_DATE

Syntax

```
Date TO_DATE(INT time)
Date TO_DATE(VARCHAR date)
Date TO_DATE(VARCHAR date, VARCHAR format)
```

Input parameters

- time: specifies the number of days from January 1, 1970 to a specified date. This parameter is of the INT data type.
- date: specifies a date of the VARCHAR data type. The default format is yyyy-MM-dd.
- format: specifies a date format of the VARCHAR data type.

Description

Converts a date of the INT or VARCHAR data type into a date of the DATE data type.

Example

Input example

date1 (bigint)	date2 (VARCHAR)	date3 (VARCHAR)
100	2017-09-15	20170915

Sample code

```
SELECT  TO_DATE(date1) as var1,
    TO_DATE(date2) as var2,
    TO_DATE(date3, "yyyyMMdd") as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
1970-04-11	2017-09-15	2017-09-15

3.1.11.3.16. TO_TIMESTAMP

Syntax

```
TIMESTAMP TO_TIMESTAMP(BIGINT time)
TIMESTAMP TO_TIMESTAMP(VARCHAR date)
TIMESTAMP TO_TIMESTAMP(VARCHAR date, VARCHAR format)
```

Arguments

- time: specifies a time value of the BIGINT type. The unit is milliseconds.
- date: specifies a date of the VARCHAR type. The default format is yyyy-MM-dd HH:mm:ss[. SSS].
- format: specifies the source date format of the VARCHAR type.

Description

Convert a date of the BIGINT or VARCHAR type to a date of the TIMESTAMP type.

Examples

Input example

timestamp1 (bigint)	timestamp2 (VARCHAR)	timestamp3 (VARCHAR)
1513135677000	2017-09-15 00:00:00	20170915000000

Sample code

```
SELECT TO_TIMESTAMP(timestamp1) as var1,
    TO_TIMESTAMP(timestamp2) as var2,
    TO_TIMESTAMP(timestamp3, "yyyyMMddHHmmss") as var3
FROM T1
```

Output example

var1 (TIMESTAMP)	var2 (TIMESTAMP)	var3 (TIMESTAMP)
2017-12-13 03:27:57.0	2017-09-15 00:00:00.0	2017-09-15 00:00:00.0

3.1.11.3.17. UNIX_TIMESTAMP

Syntax

```
BIGINT UNIX_TIMESTAMP()
BIGINT UNIX_TIMESTAMP(VARCHAR date)
BIGINT UNIX_TIMESTAMP(VARCHAR date, VARCHAR format)
```

Arguments

- date: specifies a date of the VARCHAR type. The default format is yyyy-MM-dd HH:mm:ss.
- format: specifies the source date format of the VARCHAR type. The default format is yyyy-MM-dd HH:mm:ss.

Description

Convert the specified date to a Unix timestamp (in seconds) of the BIGINT type. Both the date and format parameters are optional. If neither is specified, the Unix timestamp (in seconds) of the current time is returned. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

nullstr (VARCHAR)
null

Sample code

```
SELECT  UNIX_TIMESTAMP() as big1,
    UNIX_TIMESTAMP(nullstr) as big2
FROM T1
```

Output example

big1 (BIGINT)	big2 (BIGINT)
1403006911	null

3.1.11.3.18. WEEK

Syntax

```
INT WEEK(DATE date)
INT WEEK(TIMESTAMP time)
```

Arguments

- date: specifies a date of the DATE type.
- time: specifies a date of the TIMESTAMP type.

Description

Calculate the week number of the specified date in the year. The valid week number range is from 1 to 53.

Examples

Input example

dateStr(VARCHAR)	date1 (DATE)	ts1 (TIMESTAMP)
2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT WEEK(TIMESTAMP '2017-09-15 00:00:00') as int1,
    WEEK(date1) as int2, WEEK(ts1) as int3, WEEK(CAST(dateStr AS DATE)) as int4
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)
37	45	41	37

3.1.11.3.19. YEAR

Syntax

```
INT YEAR(TIMESTAMP time)
INT YEAR(DATE date)
```

Arguments

- date: specifies a date of the DATE type.
- time: specifies a date of the TIMESTAMP type.

Description

Extract the year from the specified date.

Examples

Input example

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate (DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT YEAR(TIMESTAMP '2016-09-15 00:00:00') as int1,
    YEAR(DATE '2017-09-22') as int2,
    YEAR(tdate) as int3,
    YEAR(ts) as int4,
    YEAR(CAST(dateStr AS DATE)) as int5,
    YEAR(CAST(tsStr AS TIMESTAMP))  as int6
FROM T1
```

Output example

int1 (BIGINT)	int2 (BIGINT)	int3 (BIGINT)	int4 (BIGINT)	int5 (BIGINT)	int6 (BIGINT)
2016	2017	2017	2017	2015	2017

3.1.11.4. Conditional functions

3.1.11.4.1. CASE WHEN

Syntax

```
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
```

Input parameters

Parameter	Data type
a	BOOLEAN
b	BOOLEAN
c	BOOLEAN
d	BOOLEAN
e	BOOLEAN

Description

- If a is true, b is returned.
- If a is false and c is true, d is returned.
- If both a and c are false, e is returned.

Example

Note

If the CASE WHEN function returns a constant string, spaces are added after the string. An example is described as follows. When the return value is "ios", several spaces are added after this string.

```
case when device_type = 'android'
then 'android'
else 'ios'
end as os
```

You can resolve the issue in the following two ways:

- Use the trim() method to remove the spaces. As shown in the following example, you can change os to trim(os) in the statement.
- Use the cast() method to convert the constant string into the VARCHAR data type.

- Input example

device_type (VARCHAR)
android
ios
win

- Sample code 1: using the trim() method

```
SELECT
  trim(os), // The trim() method is used.
  CHAR_LENGTH(trim(os)) // The trim() method is used.
from(
  SELECT
    case when device_type = 'android'
    then 'android'
    else 'ios'
end as os
FROM T1
);
```

Sample code 2: using the cast() method

```
SELECT
 os,
CHAR_LENGTH(os)
from
(SELECT
 case when device_type = 'android'
 then cast('android' as varchar) // The cast() method is used.
 else cast('ios' as varchar) // The cast() method is used.
end as os
FROM T1
);
```

- Output example

os (VARCHAR)	length (INT)
android	7
ios	3
ios	3

3.1.11.4.2. COALESCE

Syntax

```
COALESCE(A,B,...)
```

Arguments

A and B: specify values to be checked. NULL values are allowed in the list of values. Non-NULL values in the list must be of the same type. Otherwise, an exception occurs. The return type is the same as the argument type.

Description

Return the first non-NULL value in the specified list. If all values in the list are NULL, the return value is NULL.

> **Note** The list must contain at least one argument. Otherwise, an exception occurs.

Examples

Input example

var1(VARCHAR)	var2(VARCHAR)
null	30

Sample code

```
SELECT COALESCE(var1,var2) as aa
FROM T1;
```

Output example

aa(VARCHAR)
30

3.1.11.4.3. IF

Syntax

```
T IF(BOOLEAN testCondition, T valueTrue, T valueFalseOrNull)
```

> **Note** T indicates any return type.

Arguments

- testCondition: BOOLEAN type
- valueTrue and valueFalseOrNull: These two arguments must be of the same type. The specific type is not restricted.

Description

Use the BOOLEAN value of testCondition as the criterion. If the value is true, valueTrue is returned. If the value is false, valueFalseOrNull is returned. If testCondition is NULL, valueFalseOrNull is returned. If any other argument is NULL, the operation is performed based on normal semantics, and the return value is of the T type.

Examples

Input example

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	Jack	Harry
1	2	Jack	null
1	2	null	Harry

Sample code

```
SELECT IF(int1 < int2,str1, str2) as int1
FROM T1
```

Output example

int1(VARCHAR)
Jack
Jack
null

3.1.11.4.4. IS_ALPHA

Syntax

```
BOOLEAN IS_ALPHA(VARCHAR str)
```

Arguments

str: specifies a string of the VARCHAR type.

Description

Check whether the specified string contains only letters. If it contains only letters, TRUE is returned. Otherwise, FALSE is returned.

Examples

Input example

e (VARCHAR)	f (VARCHAR)	g (VARCHAR)
3	asd	null

Sample code

```
SELECT  IS_ALPHA(e) as boo1,IS_ALPHA(f) as boo2,IS_ALPHA(g) as boo3
FROM T1
```

Output example

boo1 (BOOLEAN)	boo2 (BOOLEAN)	boo3 (BOOLEAN)
false	Yes	false

3.1.11.4.5. IS_DECIMAL

Syntax

```
BOOLEAN IS_DECIMAL(VARCHAR str)
```

Input parameter

str: specifies a string of the VARCHAR data type.

Description

If a specified string contains only decimal characters, true is returned. Otherwise, false is returned.

Example

Input example

a (VARCHAR)	b (VARCHAR)	c (VARCHAR)	d (VARCHAR)	e (VARCHAR)	f (VARCHAR)	g (VARCHAR)
1	123	2	11.4445	3	asd	null

Sample code

```
SELECT IS_DECIMAL(a) as boo1,IS_DECIMAL(b) as boo2,IS_DECIMAL(c) as boo3,
    IS_DECIMAL(d) as boo4,IS_DECIMAL(e) as boo5,IS_DECIMAL(f) as boo6,IS_DECIMAL(g) as boo7
FROM T1
```

Output example

boo1 (BOOLEAN)	boo2 (BOOLEAN)	boo3 (BOOLEAN)	boo4 (BOOLEAN)	boo5 (BOOLEAN)	boo6 (BOOLEAN)	boo7 (BOOLEAN)
true	true	true	true	true	false	false

3.1.11.4.6. IS_DIGIT

Syntax

```
BOOLEAN IS_DIGIT(VARCHAR str)
```

Arguments

str: specifies a string of the VARCHAR type.

Description

Check whether the specified string contains only digits. If it only contains digits, TRUE is returned. Otherwise, FALSE is returned. The return value is of the BOOLEAN type.

Examples

Input example

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

Sample code

```
SELECT  IS_DIGIT(e) as boo1,IS_DIGIT(f) as boo2,IS_DIGIT(g) as boo3
FROM T1
```

Output example

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
true	false	false

3.1.11.4.7. NULLIF

Syntax

```
NULLIF(A,B)
```

Arguments

A and B: INT type

Description

If A and B have the same value, null is returned. If A and B have different values, the value of the first argument is returned.

Examples

Input example

var1(INT)	var2(INT)
30	30

Sample code

```
SELECT NULLIF(var1,var2) as aa
FROM T1;
```

Output example

aa(VARCHAR)
null

3.1.11.5. Table-valued functions

3.1.11.5.1. GENERATE_SERIES

Syntax

```
GENERATE_SERIES(INT from, INT to)
```

Arguments

- from: specifies a lower limit of the INT type.
- to: specifies an upper limit of the INT type.

Description

Generate a series of continuous values from the lower limit to the upper limit minus one.

Examples

Input example

s(INT)	e(INT)
1	3
-2	1

Sample code

```
SELECT s, e, v
FROM T1, lateral table(GENERATE_SERIES(s, e))
as T(v)
```

Output example

s(INT)	e(INT)	v(INT)
1	3	1
1	3	2
-2	1	-2
-2	1	-1
-2	1	-0

3.1.11.5.2. JSON_TUPLE

Syntax

```
JSON_TUPLE(str, path1, path2 ..., pathN)
```

Arguments

- str: specifies a JSON string.
- path1 to pathN: specify strings that represent paths, which do not need to begin with a dollar sign ($).

Description

Obtain the value represented by each path string from a JSON string.

Examples

Input example

d(VARCHAR)	s(VARCHAR)
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	qwe3
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	qwe2

Sample code

```
SELECT d, v
FROM T1, lateral table(JSON_TUPLE(d, 'qwe', s))
as T(v)
```

Output example

d(VARCHAR)	v(VARCHAR)
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	asd
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	asd3
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	asd4
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	asd5

3.1.11.5.3. STRING_SPLIT

Syntax

```
STRING_SPLIT(VARCHAR , separator)
```

Arguments

separator: specifies a separator of the VARCHAR type.

Description

Use a separator to split a string into multiple substrings.

Examples

Input example

d (VARCHAR)	s (VARCHAR)
abc-bcd	-
hhh	-

Sample code

```
SELECT d, v
FROM T1, lateral table(STRING_SPLIT(d, s))
as T(v)
```

Output example

d (VARCHAR)	v (VARCHAR)
abc-bcd	abc
abc-bcd	bcd
hhh	hhh

3.1.11.6. Type conversion functions

3.1.11.6.1. CAST

Syntax

```
CAST(A AS type)
```

Argument

A: For more information about the argument type, see Data types.

Description

Convert a value to a specified type.

Examples

Input example

var1 (VARCHAR)	var2 (INT)
1000	30

Sample code

```
SELECT CAST(var1 AS INT) as aa
FROM T1;
```

Output example

aa (INT)
1000

3.1.11.7. Aggregate functions

3.1.11.7.1. AVG

Syntax

```
AVG(A)
```

Argument

A: specifies a value of numeric types including TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE.

Description

Return the average number.

Examples

Input example

var1 (INT)	var2 (INT)
4	30
6	30

Sample code

```
SELECT AVG(var1) as aa
FROM T1;
```

Output example

aa (INT)
5

3.1.11.7.2. CONCAT_AGG

Syntax

```
CONCAT_AGG([linedelimiter, ] value)
```

Argument

linedelimiter: specifies a connector, which currently can only be a string constant.

Description

Use a connector to concatenate multiple fields. The default connector is "\n". The return value is a new concatenated string of the VARCHAR type.

Examples

Input example

c (VARCHAR)
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi

Sample code

```
SELECT concat_agg(c) as var1,
    concat_agg('-', c) as var2
FROM MyTable
GROUP BY c
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)
Hi\nHi\nHi\nHi\nHi\nHi\nHi\nHi	Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi

3.1.11.7.3. COUNT

Syntax

```
COUNT(A)
```

Arguments

A

- Supported data types include numeric types (TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE), BOOLEAN, and VARCHAR.
- The DATE, TIME, TIMESTAMP, and VARBINARY data types are not supported.

Description

Return the number of input parameters.

Examples

Input example

var1 (VARCHAR)
1000
100
10
1

Sample code

```
SELECT COUNT(var1) as aa
FROM T1;
```

Output example

aa (BIGINT)
4

3.1.11.7.4. FIRST_VALUE

Syntax

```
T FIRST_VALUE(value: T)
```

To obtain the first record based on parameters, define the function as follows:

```
T FIRST_VALUE(value: T, order: Long)
```

Arguments

- value: specifies a value of any type. Only one argument is supported.
- order: specifies a record with the smallest order value as the first record.

Description

Obtain the first non-null record of a stream.

Examples

Input example

a (BIGINT)	b (INT)	c (VARCHAR)
1L	1	"Hello"
2L	2	"Hello"
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"
20L	20	"Hello World"

Sample code

```
SELECT
c,
first_value(b) OVER (PARTITION BY c ORDER BY proctime RANGE UNBOUNDED preceding) as var1
from T1
```

Output example

c (VARCHAR)	var1 (INT)
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
Hello World	7
Hello World	7
Hello World	7

3.1.11.7.5. LAST_VALUE

Syntax

```
T LAST_VALUE(value: T)
```

To return the last value in an ordered set of values, define the function as follows:

```
T LAST_VALUE(value: T, order: Long)
```

Arguments

- value: specifies a value of any type.
- order: specifies the order of the values. A value with the greatest order value is considered as the last value.

Description

Obtain the last data record of a stream.

Examples

Input example

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	"Hello"
2L	2	"Hello"

a(BIGINT)	b(INT)	c(VARCHAR)
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"
20L	20	"Hello World"

Sample code

```
SELECT
c,
last_value(b) OVER (PARTITION BY c ORDER BY proctime RANGE UNBOUNDED preceding) as var1
from T1
```

Output example

c(VARCHAR)	var1(INT)
Hello	1
Hello	2
Hello	3
Hello	4
Hello	5
Hello	6
Hello World	7
Hello World	8
Hello World	20

3.1.11.7.6. MAX

Syntax

```
MAX(A)
```

Arguments

A

- Supported data types include numeric values (TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE), BOOLEAN, and VARCHAR.
- The DATE, TIME, TIMESTAMP, and VARBINARY data types are not supported.

Description

Return the maximum value of all the input values.

Examples

Input example

var1(INT)
4
8

Sample code

```
SELECT MAX(a) as aa
FROM T1;
```

Output example

aa(INT)
8

3.1.11.7.7. MIN

Syntax

```
MIN(A)
```

Arguments

A

- Supported data types include numeric values (TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE), BOOLEAN, and VARCHAR.
- The DATE, TIME, TIMESTAMP, and VARBINARY data types are not supported.

Description

Return the minimum value of all the input values.

Examples

Input example

var1(INT)
4
8

Sample code

```
SELECT MIN(a) as aa
FROM T1;
```

Output example

aa(INT)
4

3.1.11.7.8. STDDEV_POP

Syntax

```
T STDDEV_POP(T value)
```

Argument

value: specifies a number of the BIGINT or DOUBLE type.

Description

Return the standard deviation for the population of all values in the specified expression.

Examples

Input example

a (DOUBLE)	c (VARCHAR)
0	Hi
1	Hi
2	Hi

a (DOUBLE)	c (VARCHAR)
3	Hi
4	Hi
5	Hi
6	Hi
7	Hi
8	Hi
9	Hi

Sample code

```
SELECT c, STDDEV_POP(a) as dou
FROM MyTable
GROUP BY c
```

Output example

c (VARCHAR)	dou1 (DOUBLE)
Hi	2.8722813232690143

3.1.11.7.9. SUM

Syntax

```
SUM(A)
```

Arguments

A: specifies a value of the numeric types including TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE.

Description

Return the sum of all input values.

Examples

Input example

var1 (INT)
4
4

Sample code

```
SELECT sum(a) as aa
FROM T1;
```

Output example

aa (INT)
8

3.1.11.7.10. VAR_POP

Syntax

```
T VAR_POP(T value)
```

Argument

value: specifies a number of the BIGINT or DOUBLE type.

Description

Return the statistical variance for the population of all values in the specified expression.

Examples

Input example

a (BIGINT)	c (VARCHAR)
2900	Hi
2500	Hi
2600	Hi
3100	Hello
11000	Hello

Sample code

```
SELECT
  VAR_POP(a) as `result`,
  c
FROM MyTable
GROUP BY c
```

Output example

result (BIGINT)	c
28889	Hi
15602500	Hello

3.1.11.8. Other functions

3.1.11.8.1. UUID

Syntax

```
VARCHAR UUID()
```

Arguments

None

Description

Return a globally unique identifier.

Examples

Sample code

```
SELECT  uuid() as `result`
FROM T1
```

Output example

result(VARCHAR)
a364e414-e68b-4e5c-9166-65b3a153e257

3.1.12. UDXs

3.1.12.1. Overview

You can extend the following three types of user-defined functions (UDFs):

- User-defined scalar function (UDSF)

  A UDSF can be used in a SQL expression and returns only one value.

- User-defined aggregation function (UDAF)

  A UDAF aggregates multiple scalar values into one value.

- User-defined table function (UDTF)

  A UDTF takes zero, one, or multiple scalar values as input parameters and returns any number of rows. Among UDFs, UDTFs are the only type that can return multiple fields.

Download JAR files

- For Realtime Compute versions earlier than 2.0, download the following JAR files:
  - flink-streaming-java_2.11
  - flink-table_2.11

  > **Note** After the JAR files are downloaded, you must modify the dependency information in the `pom.xml` file as follows:

```
<dependency>&amp;nbsp;
   <groupId>com.alibaba.blink</groupId>
   <artifactId>flink-table_2.11</artifactId>
   <version>blink-1.4-SNAPSHOT</version>
   <scope>provided</scope>
 </dependency>
 <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java_2.11</artifactId>
    <version>blink-1.4-SNAPSHOT</version>
    <scope>provided</scope>
 </dependency>
```

- For Realtime Compute version 2.0 and later, download the following JAR files:
  - flink-streaming-java_2.11
  - flink-table_2.11
  - flink-core-blink-2.2.4

Note After the JAR files are downloaded, you must modify the pom.xml file as follows:

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-core</artifactId>
  <version>blink-2.2.4-SNAPSHOT</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-table_2.11</artifactId>
  <version>blink-2.2.4-SNAPSHOT</version>
  <scope>provided</scope>
</dependency>
<dependency>
   <groupId>org.apache.flink</groupId>
   <artifactId>flink-streaming-java_2.11</artifactId>
   <version>blink-2.2.4-SNAPSHOT</version>
   <scope>provided</scope>
</dependency>
```

Register and use a UDF

1. Log on to the Realtime Compute console. For more information, see **Log on to the Realtime Compute console** in User Guide.
2. Click **Development** in the top navigation bar to go to the development page.
3. Click **Resources** in the left-side navigation pane.
4. On the **Resources** page, click **Create Resource** in the upper right corner to go to the **Upload Resource** dialog box.
5. Specify settings for a resource as follows.

Field	Description
Location	Currently, you can only upload local resources.
Resource	Click **Upload Resource** to select a resource to be referenced.
Resource Name	Enter a name for the resource.
Resource Description	Enter the resource description.
Resource Type	Select a type for a resource: JAR, DICTIONARY, or PYTHON.

6. On the **Resources** page, locate the new resource, and move the pointer over **More** in the Actions column.
7. Select **Reference** from the drop-down list to load the resource at runtime.
8. In the code editor, enter a declaration for the user-defined function at the beginning. An example is described as follows:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

3.1.12.2. User-defined scalar function (UDSF)

This topic describes how to create a UDSF for Realtime Compute.

Definition

A UDSF maps zero, one, or multiple scalar values to a new scalar value.

Build the development environment

For more information, see Download JAR files.

Write code that implements business logic

To define a scalar function, you must extend the ScalarFunction class by implementing one or more `eval()` methods. The `open()` and `close()` methods are optional. An example of Java code is provided as follows:

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
  // The open() method is optional.
  // To write the open() method, you must import org.apache.flink.table.functions.FunctionContext.
  @Override
  public void open(FunctionContext context) {
    }
  public long eval(String a) {
    return a == null ? 0 : a.length();
  }
  public long eval(String b, String c) {
    return eval(b) + eval(c);
  }
  // The close() method is optional.
  @Override
  public void close() {
    }
}
```

Publish and use a UDSF

For more information about how to publish and use a UDSF, see **Publish** in User Guide.

Sample code of a UDSF

```
-- udf str.length()
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
create table sls_stream(
  a int,
  b int,
  c varchar
) with (
  type='sls',
  endPoint='yourEndpoint',
  accessKeyId='yourAccessId',
  accessKeySecret='yourAccessSecret',
  startTime = '2017-07-04 00:00:00',
  project='yourProjectName',
  logStore='yourLogStoreName',
  consumerGroup='consumerGroupTest1'
);
create table rds_output(
  id int,
  len bigint,
  content VARCHAR
) with (
  type='rds',
  url='yourDatabaseURL',
  tableName='yourDatabaseTableName',
  userName='yourDatabaseUserName',
  password='yourDatabasePassword'
);
insert into rds_output
select
  a,
  stringLengthUdf(c),
  c as content
from sls_stream
```

FAQ

Q: Why does a UDSF that generates random numbers always returns the same value?

A: If no parameters are defined for a UDSF and you do not declare the function as nondeterministic, the function may be optimized during compilation to return a constant value. To avoid this, you can override the `isDeterministic()` method to make it return `false`.

3.1.12.3. User-defined aggregation function (UDAF)

This topic describes how to create a UDAF for Realtime Compute.

Definition

A UDAF aggregates multiple values into a single value.

Methods of a UDAF

The following code describes the core methods of a UDAF.

> **Note** A UDAF can be implemented in Java or Scala. We recommend that you use Java because Scala data types may cause unnecessary overhead when calling functions.

- createAccumulator() and getValue() methods

```
/*
* @param <T> The type of the aggregation result returned by a UDAF.
* @param <ACC> The type of a UDAF accumulator. An accumulator stores the intermediate computing res
ults of a UDAF. You can design an accumulator for each UDAF as required.
*/
public abstract class AggregateFunction<T, ACC> extends UserDefinedFunction {
/*
* Initializes the accumulator of the AggregateFunction.
* Calls the following method once when aggregation is performed for the first time.
*/
public ACC createAccumulator();
/*
* Calls the following method after each aggregation is complete.
*/
public T getValue(ACC accumulator);
}
```

> **Note**
> - The createAccumulator() and getValue() methods can be defined in the AggregateFunction abstract class.
> - A UDAF must contain at least one accumulate() method.

- accumulate() method

```
public void accumulate(ACC accumulator, ...[ User-specified parameters]...) ;
```

Note

- You need to implement an accumulate() method to process input data and update the provided accumulator with the computing result.
- The first parameter of the accumulate() method must be an accumulator of the ACC type as defined in the AggregateFunction class. During processing, you can use the underlying runtime code to send the previous accumulator and the specified upstream data to the operator for calculation. No limits are set for the data types or the number of data records.

- retract() and merge() methods

  The createAccumulator(), getValue(), and accumulate() methods are mandatory for each UDAF. The retract() and merge() methods are also needed in certain Realtime Compute scenarios.

  In many Realtime Compute scenarios, computing is an early firing for an infinite stream. To refine the early fired results, you need to implement a retract() method. The SQL translation optimizer automatically determines the circumstances under which input data needs to be retracted. It also automatically determines the operations you need to process the data with the retraction labels. You must implement a retract() method to retract input data.

  ```
  public void retract(ACC accumulator, ...[ User-specified parameters]...) ;
  ```

  Note

  - The retract() method is a reverse operation of the accumulate() method. For example, in a count UDAF, the computing result increases by 1 when the accumulate() method is called to process a data record. The result decreases by 1 when the retract() method is called to process a data record.
  - Similar to the accumulate() method, the first parameter of the retract() method must be an accumulator of the ACC type as defined in the aggregate function class. During data processing, the runtime code sends the previous value in the accumulator and the specified upstream data to the operator for calculation. No limits are set for the data types or the number of data records.

  The merge() method is required in certain Realtime compute scenarios. For example, when you use session windows to aggregate data, you need to implement the merge() method. Realtime Compute can handle out-of-order data. Newly arrived data may fill the gap between two separate sessions, which results in the merge of the two sessions. In this case, you need to implement the merge() method to integrate multiple accumulators into one accumulator.

  ```
  public void merge(ACC accumulator, Iterable<ACC> its);
  ```

> Note
>
> - The first parameter of the merge() method must be an accumulator of the ACC type as defined in the AggregateFunction class. After the merge() method is complete, the state of the AggregateFunction is stored in the first accumulator.
> - The second parameter of the merge() method is an iterator of one or more ACC-type accumulators.

Build the development environment

For more information about how to build the development environment, see Download JAR files.

Write code that implements business logic

An example of Java code is provided as follows:

```
import org.apache.flink.table.functions.AggregateFunction;
public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
  // Defines the data schema of the accumulator that stores the state of the count UDAF.
  public static class CountAccum {
    public long total;
  }
  // Initializes the accumulator of the count UDAF.
  public CountAccum createAccumulator() {
    CountAccum acc = new CountAccum();
    acc.total = 0;
    return acc;
  }
  // Calls the getValue() method to obtain the result of the count UDAF from the accumulator that stores the
UDAF state.
  public Long getValue(CountAccum accumulator) {
    return accumulator.total;
  }
  // Calls the accumulate() method to update the accumulator that stores the state of the count UDAF based
on input data.
  public void accumulate(CountAccum accumulator, Object iValue) {
    accumulator.total++;
  }
  public void merge(CountAccum accumulator, Iterable<CountAccum> its) {
    for (CountAccum other : its) {
      accumulator.total += other.total;
    }
  }
}
```

> **Note** For a subclass of AggregateFunction, the open() and close() methods are optional. For more information, see the examples of a UDF or UDTF.

Publish and use a UDAF

For more information about how to publish and use a UDAF, see **Publish** in User Guide.

Sample code of a UDAF

```
-- Uses a UDAF to calculate the count.
CREATE FUNCTION countUdaf AS 'com.hjc.test.blink.sql.udx.CountUdaf';
create table sls_stream(
a int,
b bigint,
c varchar
) with (
type='sls',
endPoint='yourEndpoint',
accessKeyId='yourAccessId',
accessKeySecret='yourAccessSecret',
startTime = '2017-07-04 00:00:00',
project='yourProjectName',
logStore='stream-test2',
consumerGroup='consumerGroupTest3'
);
create table rds_output(
len1 bigint,
len2 bigint
) with (
type='rds',
url='yourDatabaseURL',
tableName='yourDatabaseTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);
insert into rds_output
select
count(a),
countUdaf(a)
from sls_stream
```

3.1.12.4. User-defined table function (UDTF)

Description

Similar to a user-defined scalar function (UDSF), a UDTF uses zero, one, or multiple scalar values as input parameters. The difference is that a UDTF can return a set of values, instead of a single value. The returned set of values can consist of multiple columns and multiple rows.

Build a development environment

For information about how to build a development environment, see Download JAR files.

Write code that implements business logic

To define a UDTF, you must implement the eval() method of the TableFunction class. The open() and close() methods are optional. An example of Java code is described as follows:

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
public class SplitUdtf extends TableFunction<String> {
  // The open() method is optional. To implement the open() method, you must import the org.apache.flink.
table.functions.FunctionContext class.
  @Override
  public void open(FunctionContext context) {
    // ...
    }
  public void eval(String str) {
    String[] split = str.split("\\|");
    for (String s : split) {
      collect(s);
    }
  }
  // The close() method is optional.
  @Override
  public void close() {
    // ...
    }
}
```

Return multiple rows

A UDTF can return multiple rows. If you want a UDTF to return multiple rows, you can call the `collect()` method multiple times.

Return multiple columns

A UDTF can return multiple columns. If you want a UDTF to return multiple columns, you can declare the return value as a Tuple or Row object.

- The return value is declared as a Tuple object.

  In Realtime Compute, you can use Tuple1 to Tuple25 to define 1 to 25 columns. For example, you can use Tuple3 to enable the function to return three columns. The sample code is described as follows:

```
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.table.functions.TableFunction;
// If the return value is declared as a Tuple object, you must explicitly declare the data type of the object, s
uch as STRING, LONG, and INTEGER in this example.
public class ParseUdtf extends TableFunction<Tuple3<String, Long, Integer>> {
public void eval(String str) {
String[] split = str.split(",");
// The following code is used for your reference only. In actual scenarios, you must add code that impleme
nts verification logic.
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Tuple3<String, Long, Integer> tuple3 = Tuple3.of(first, second, third);
collect(tuple3);
}
}
```

> **Note** If the return value is declared as a Tuple object, column values cannot be null and a maximum of 25 columns are allowed.

- The returned value is declared as a Row object.

  For example, you can enable this function to return three columns. The sample code is described as follows:

```
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
public class ParseUdtf extends TableFunction<Row> {
public void eval(String str) {
String[] split = str.split(",");
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Row row = new Row(3);
row.setField(0, first);
row.setField(1, second);
row.setField(2, third);
collect(row);
}
@Override
// If the return value is declared as a Row object, you must explicitly declare the data type of the object by
overriding the getResultType() method.
public DataType getResultType(Object[] arguments, Class[] argTypes) {
return DataTypes.createRowType(DataTypes.STRING, DataTypes.LONG, DataTypes.INT);
}
}
```

> **Note** If the return value is declared as a Row object, column values can be null. If the return value is declared as a Row object, you must override the `getResultType()` method.

SQL syntax

A UDTF supports CROSS JOIN and LEFT JOIN operations. To perform JOIN operations, you must use the `lateral` and `table` keywords in SQL statements. You must also specify an alias for a UDTF. For example, you can specify an alias for the preceding `ParseUdtf` function:

```
CREATE FUNCTION parseUdtf AS 'com.alibaba.blink.sql.udtf.ParseUdtf';
```

CROSS JOIN: Each row in the left table is joined with a row of data that is generated by the UDTF. If the UDTF does not generate any data for a row, the row is not returned.

```
select S.id, S.content, T.a, T.b, T.c
from input_stream as S,
lateral table(parseUdtf(content)) as T(a, b, c);
```

LEFT JOIN: Each row in the left table is joined with a row of data that is generated by the UDTF. If the UDTF does not generate any data for a row, the UDTF fields in the row are filled with null.

> **Note** When you perform LEFT JOIN operations for a UDTF, you must end the statement with an `on true` join condition.

```
select S.id, S.content, T.a, T.b, T.c
from input_stream as S
left join lateral table(parseUdtf(content)) as T(a, b, c) on true;
```

Publish and use a UDTF

For information about how to publish and use a UDTF, see the "Publish an SQL file" section in *Realtime Compute User Guide*.

Sample code

```
-- UDTF str.split("\\|");
create function splitUdtf as 'com.hjc.test.blink.sql.udx.SplitUdtf';
create table sls_stream(
a INT,
b BIGINT,
c VARCHAR
) with (
type='sls',
endPoint='<yourEndpoint>',
accessKeyId='<yourAccessKeyId>',
accessKeySecret='<yourAccessSecret>',
startTime = '2017-07-04 00:00:00',
project='<yourProjectName>',
logStore='<yourLogStoreName>',
consumerGroup='consumerGroupTest2'
);
-- Use the splitUdtf function to extract data from the c field. The table T with multiple rows and one column i
s returned. The column is named s.
create view v1 as
select a,b,c,s
from sls_stream,
lateral table(splitUdtf(c)) as T(s);
create table rds_output(
id INT,
len BIGINT,
content VARCHAR
) with (
type='rds',
url='yourDatabaseURL',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>'
);
insert into rds_output
select
a,b,s
from v1
```

3.1.12.5. Create a user-defined function (UDF) by using IntelliJ IDEA

This topic takes a Java package named com.hjc.test.blink.sql.udx as an example to describe how to use IntelliJ IDEA to create a UDF for Realtime Compute.

> **Note** This example uses IntelliJ IDEA on macOS. You need to download and install IntelliJ IDEA on your Mac before creating user-defined functions.

Preparations

1. Download Maven.
   i. Visit the Apache Maven website and download *apache-maven-3.5.3-bin.tar.gz*.
   ii. Decompress the downloaded package to a directory, for example, */Users/*****/Documents/maven*.
2. Configure environment variables.

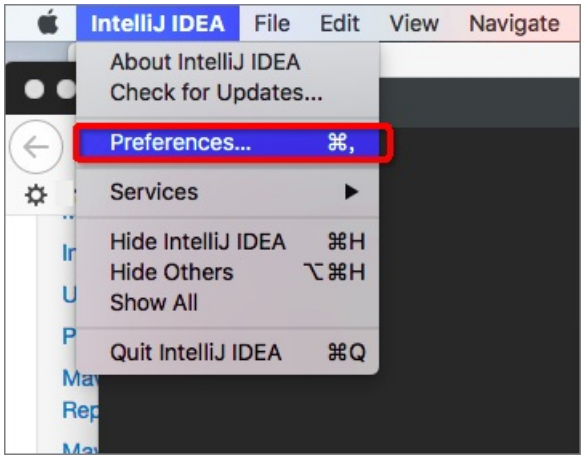
   Open Terminal on your Mac and perform the following operations:
   i. Run the **vim ~/.bash_profile** command to open the .bash_profile file.
   ii. Enter the following commands in the file: **export M2_HOME=/Users/****/Documents/maven/apache-maven-3.5.3** and **export PATH=$PATH:$M2_HOME/bin**.
   iii. Run the **:wq** command to save the configurations and exit. Run the following command in Terminal to apply the configurations: **source ~/.bash_profile**.
3. Check whether the configurations take effect.

   Run the **mvn -v** command in Terminal. If the command returns the following result, the configurations have taken effect.

   ```
   Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-04T03:39:06+08:00)
   Maven home: /Users/****/Documents/maven/apache-maven-3.5.0
   Java version: 1.8.0_121, vendor: Oracle Corporation
   Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/jre
   Default locale: zh_CN, platform encoding: UTF-8
   OS name: "mac os x", version: "10.12.6", arch: "x86_64", family: "mac"
   ```
4. Download the required JAR files to a local repository. For more information about the download URL, see Download JAR files.

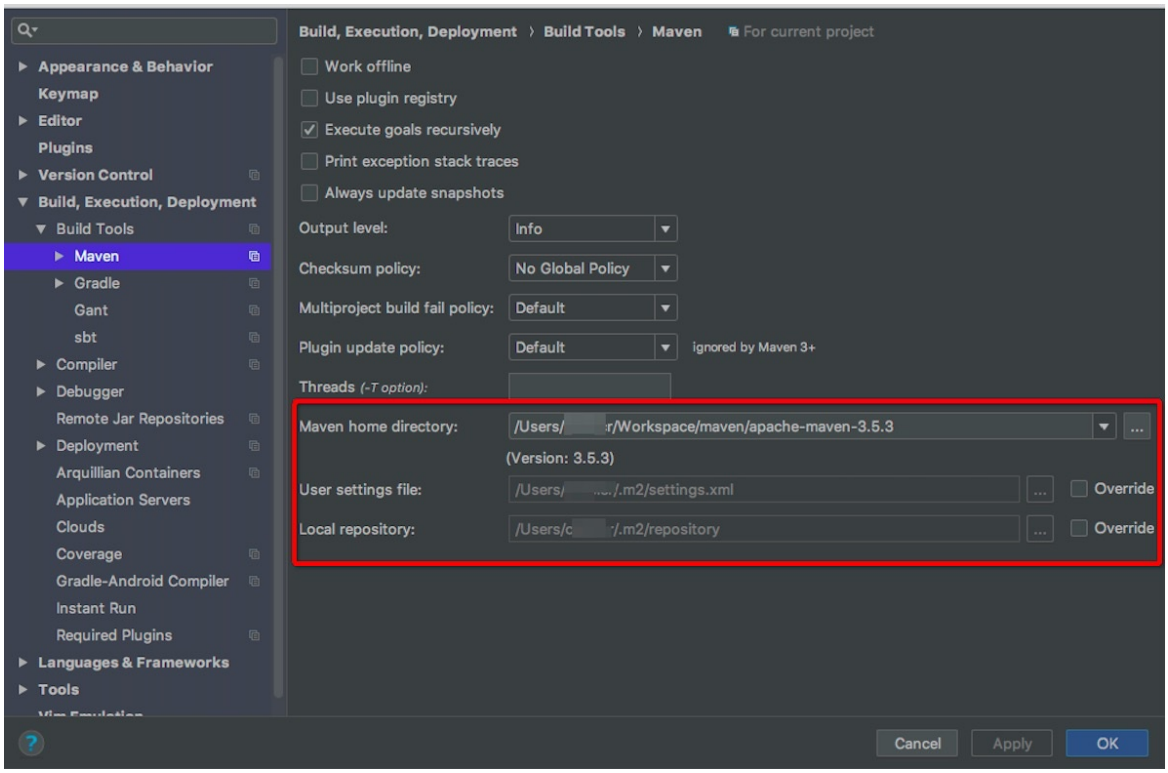
Configure Maven in IntelliJ IDEA

1. Click **IntelliJ IDEA** in the top navigation bar.
2. Select **Preferences** from the drop-down list.



3. Choose **Maven > Maven home directory** to configure the Maven home directory.

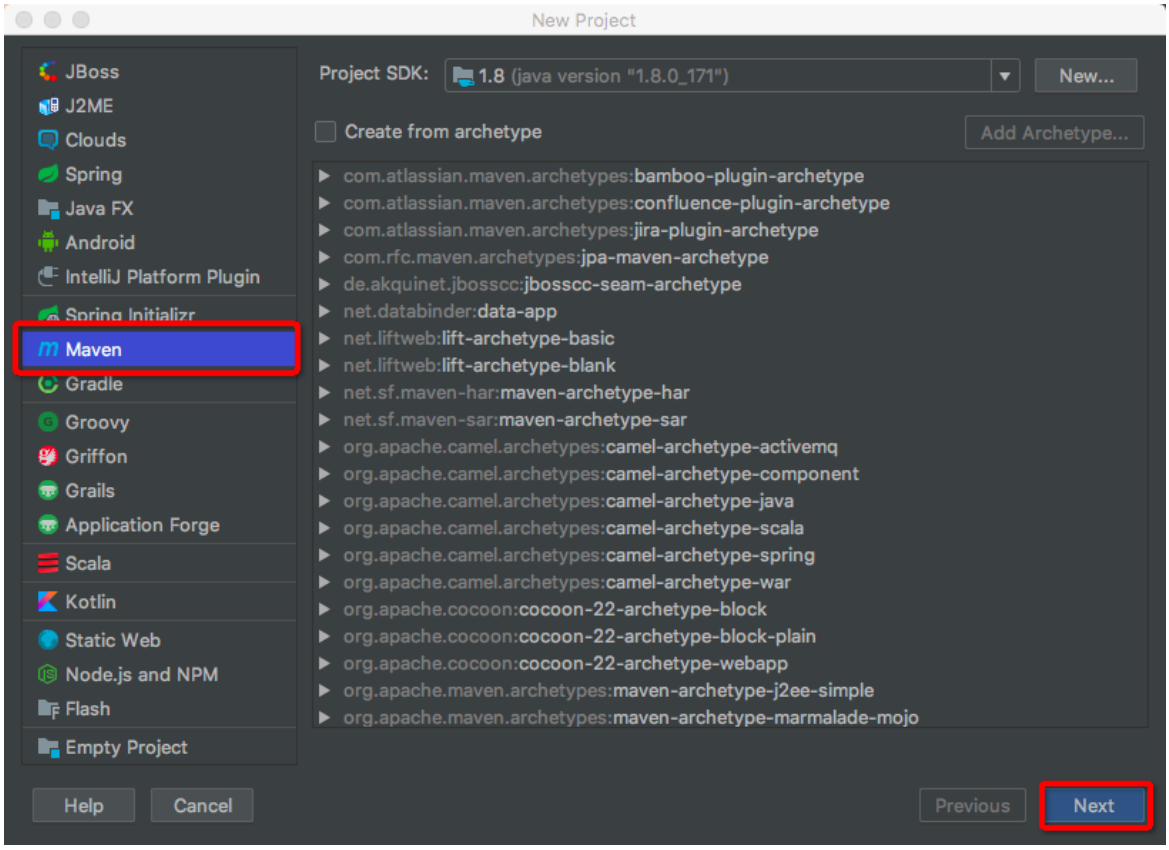
> Note You can find the Maven home directory `Maven home:/Users/****/Documents/maven/apache-maven-3.5.0` from the output of the **mvn -v** command. For more information, see Preparations.



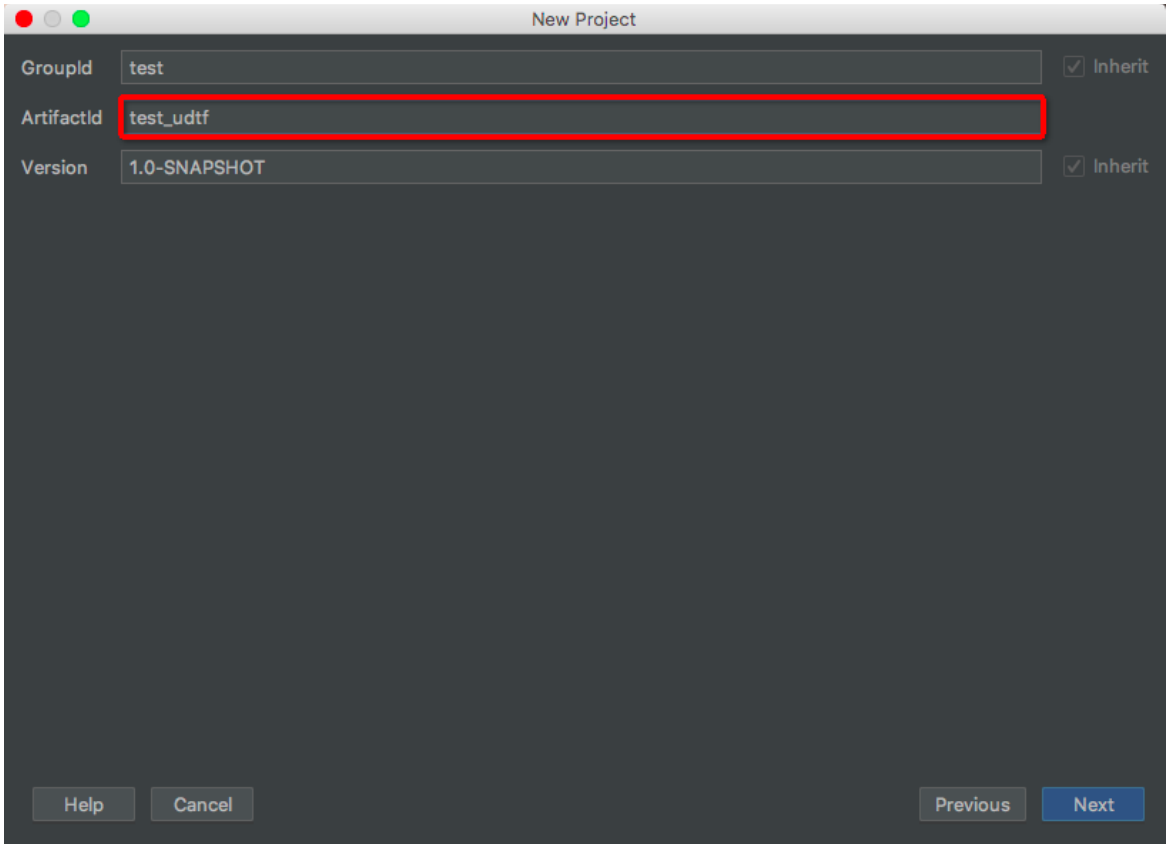
Create a Maven project

1. Click Create New Project on the home page of IntelliJ IDEA.

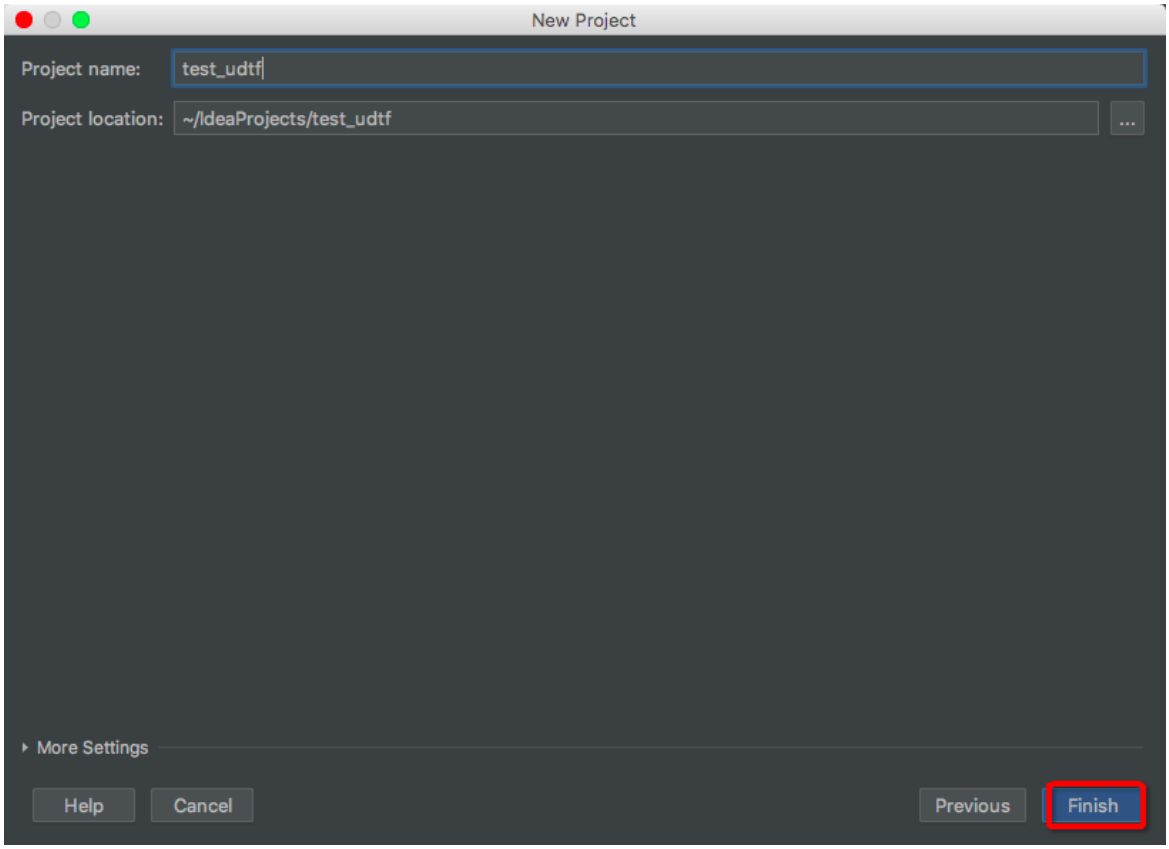
2. Select Maven.



3. Enter a project name in the ArtifactId field.

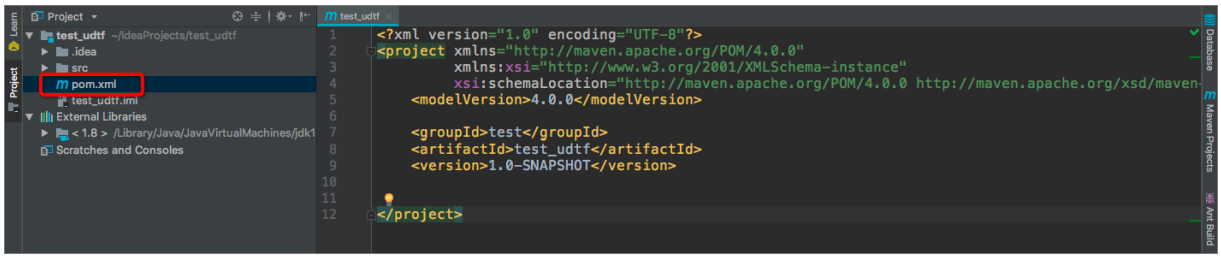


4. Click **Finish**.



Configure dependencies

Open the *pom.xml* file, and enter the following code to declare dependencies in the code editor:



- For Realtime Compute versions earlier than 2.0

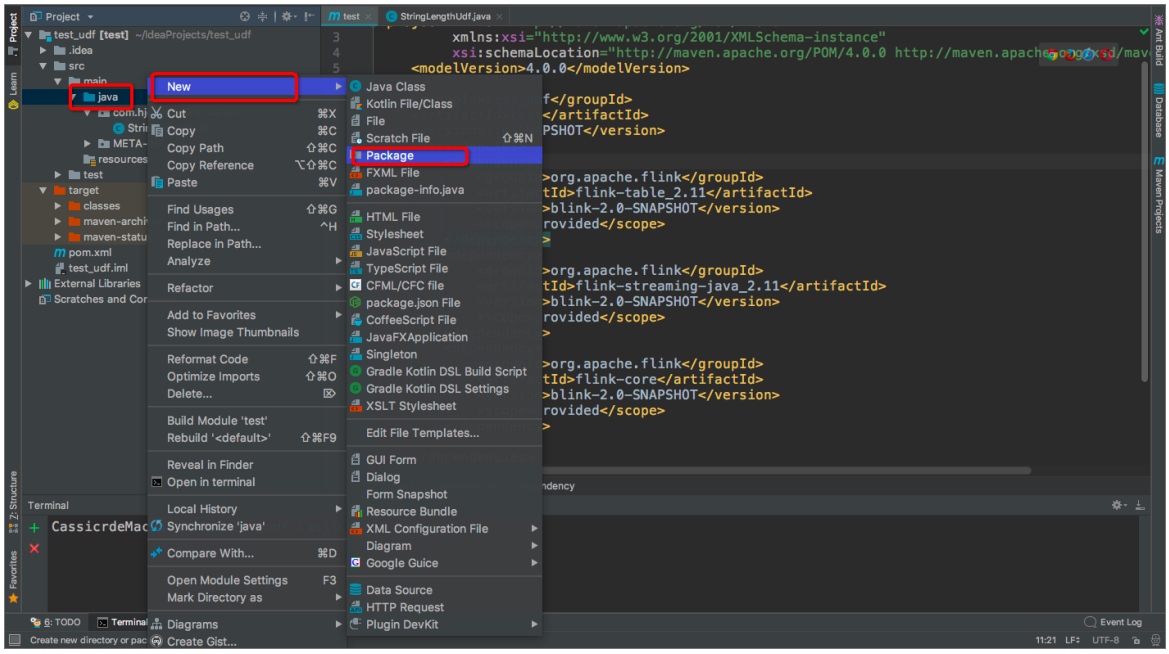
```
<dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>flink-table_2.11</artifactId>
    <version>blink-1.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-streaming-java_2.11</artifactId>
     <version>blink-1.4-SNAPSHOT</version>
     <scope>provided</scope>
  </dependency>
```

- For Realtime Compute version 2.0 and later

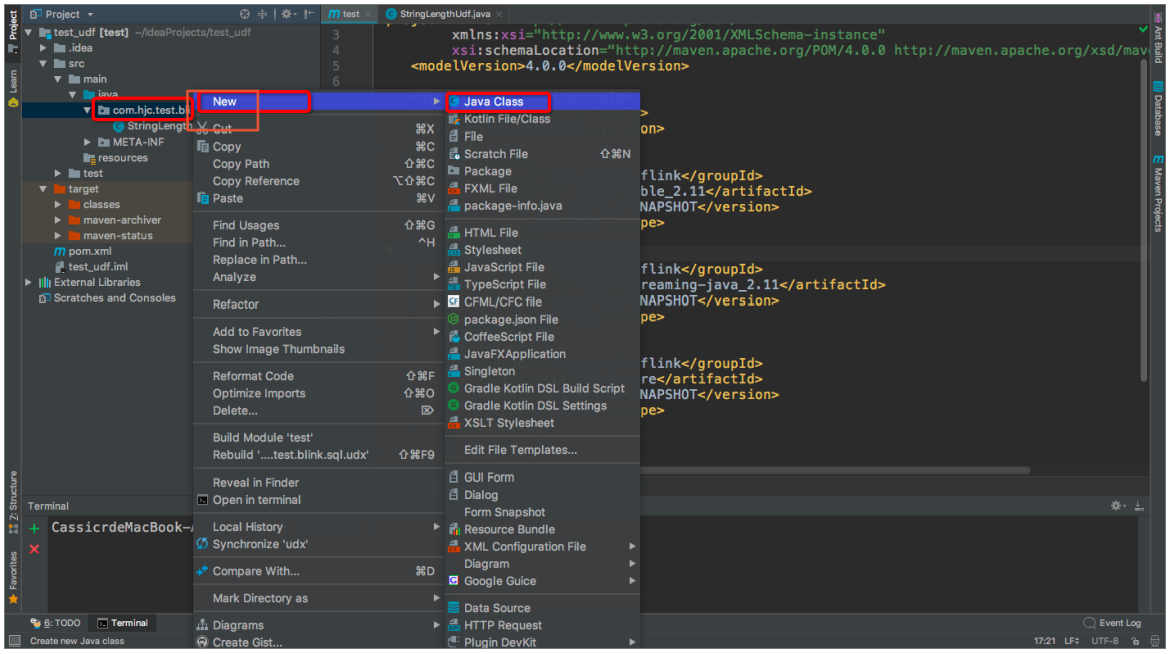
```
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-core</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table_2.11</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
     <groupId>org.apache.flink</groupId>
     <artifactId>flink-streaming-java_2.11</artifactId>
     <version>blink-2.2.4-SNAPSHOT</version>
     <scope>provided</scope>
  </dependency>
```

Test a UDF

1. Create a new package.



2. Create a new Java class.



3. Copy and paste the UDF code you have written to the Java class, and run the code to start the test. An example of UDF code is provided as follows:

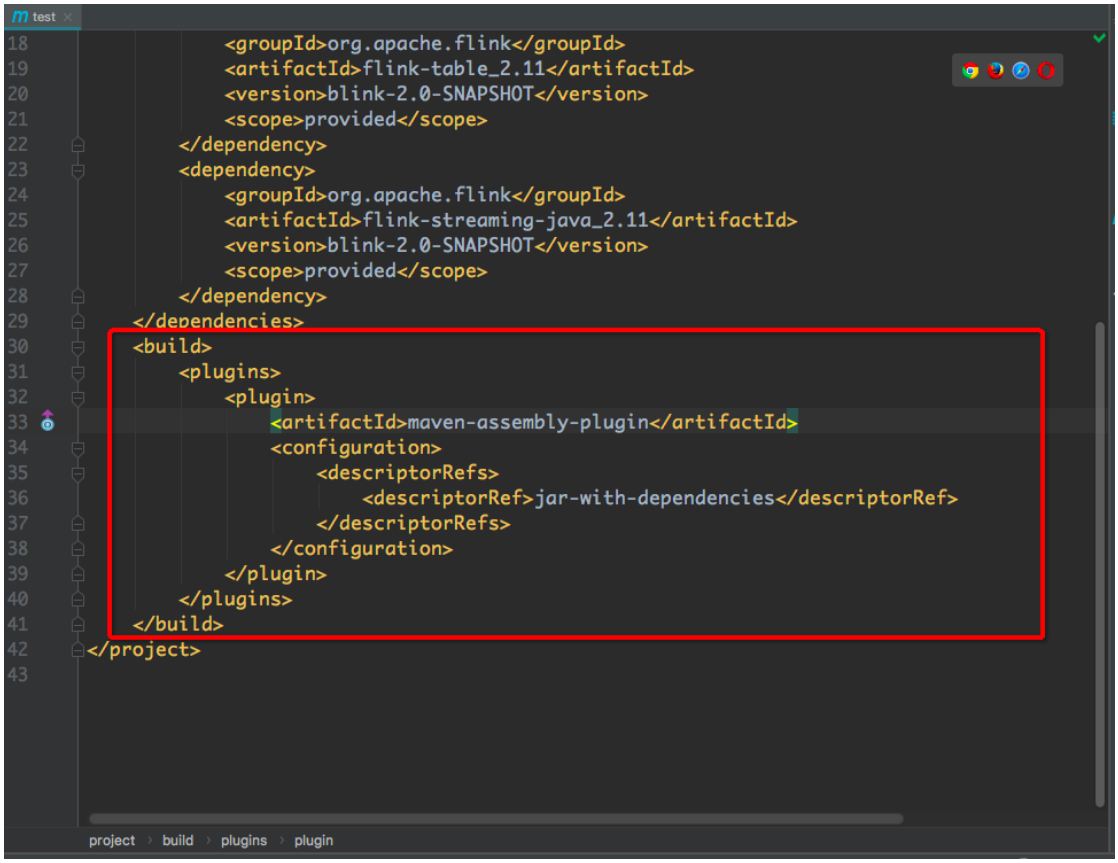
```
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
      // The open() method is optional.
       // To write the open() method, you must import org.apache.flink.table.functions.FunctionContext;'
.
@Override
public void open(FunctionContext context) {
// ...
   }
public long eval(String a) {
return a == null ? 0 : a.length();
}
public long eval(String b, String c) {
return eval(b) + eval(c);
}
// The close() method is optional.
@Override
Public void close (){
// ...
    }
}
```

4. Package the project into a JAR file.

> **Note** If your Maven project has other dependencies in addition to the Realtime Compute SDK, you need to package all JAR files in your project. Then, you can export the project as a JAR file.

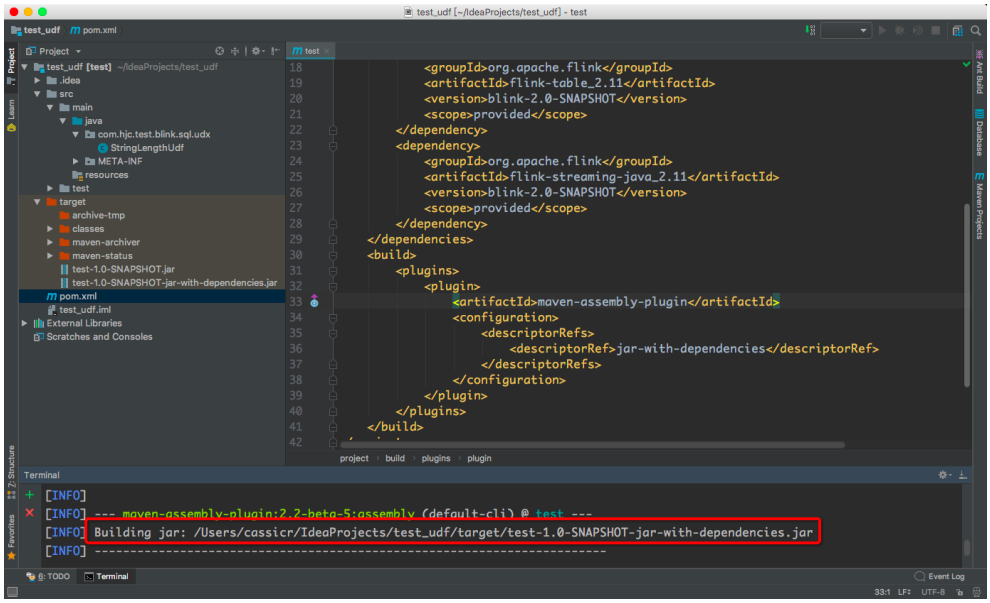
i. Enter the following code in the pom.xml file:

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
       <descriptorRefs>
        <descriptorRef>jar-with-dependencies</descriptorRef>
       </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```



ii. In the **Terminal** section at the bottom of the page, enter **mvn assembly:assembly**.

The JAR file path is generated, as shown in the following figure.



5. You can reference the JAR file in a Realtime Compute job. For more information, see Register and use a UDF.

4.E-MapReduce (EMR)

4.1. Prerequisites

4.1.1. Development prerequisites

This topic describes the prerequisites required for E-MapReduce (EMR) development.

- You have a basic understanding of Spark, Hadoop, and Hive. This document does not describe the development practices of Spark, Hadoop, and Hive. You can visit the Apache official website to obtain more development documentation.
- You have a basic understanding of programming languages Python, Java, and Scala. In this document, some examples in Scala are provided.
- You have a basic understanding of the development components of EMR. The open source community is welcome to contribute to the project. For example, your problem feedback, bug fixes, and new components are appreciated.

4.1.2. Configure the OSS URI to use EMR

This topic describes how to configure the OSS URI to use EMR.

OSS URI

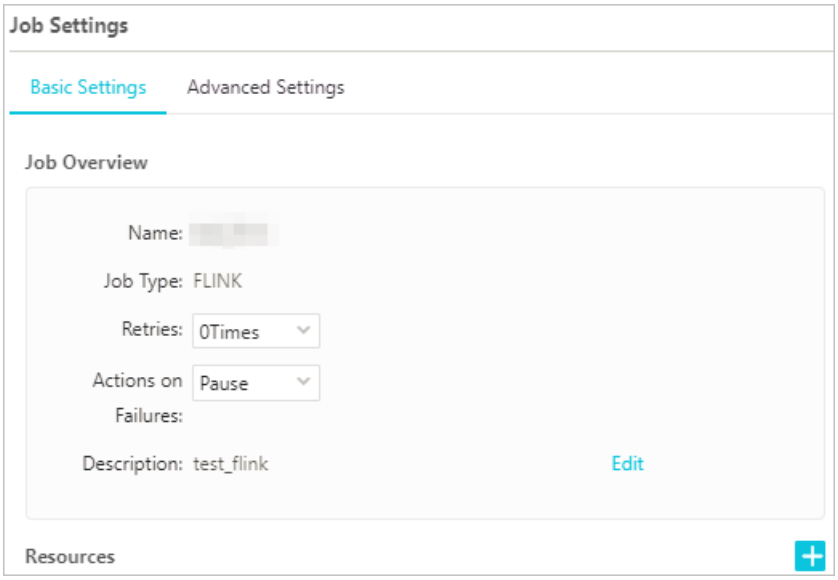
When you use EMR, you can use two types of OSS URIs:

- Native URI: *oss://[accessKeyId:accessKeySecret@]bucket[.endpoint]/object/path*

  You can use this type of URI to specify the input and output data sources for a job. This type of URI is similar to hdfs://. When you process OSS data, you can configure the AccessKey ID, AccessKey secret, and endpoint in Configuration. Alternatively, you can specify the AccessKey ID, AccessKey secret, and endpoint in the URI.

- Reference URI: *ossref://bucket/object/path*

  This type of URI is valid only in the configuration of an EMR job. It is used to specify the resources required to run the job.



The prefixes of OSS URIs, such as oss and ossref, are called schemes. Pay special attention to schemes in URIs.

> **Warning**
>
> Currently, EMR supports only **standard storage in OSS**.
>
> - EMR uses the multipart mode to upload large files to OSS. Note that if your job is interrupted, some of the result data remains in OSS. You must remove the data manually. The upload process is similar to the upload process in HDFS. The only difference is as follows: When the multipart mode is used to upload large files, file fragments are also uploaded to OSS fragment management. Therefore, you need to delete the file fragments from OSS fragment management. Otherwise, you are charged for the data storage.
> - Alternatively, you can configure the lifecycle of file fragments so that the expired fragments are automatically cleaned.

4.1.3. Use sample projects

4.1.3.1. Sample project

This sample project is a complete, compilable, and runnable project, including the sample code of MapReduce, Pig, Hive, and Spark. The details are as follows:

MapReduce

WordCount: counts the number of words.

Hive

sample.hive: queries data from a table.

Pig

sample.pig: processes OSS data instances.

Spark

- SparkPi: calculates Pi.
- SparkWordCount: counts the number of words.
- LinearRegression: linear regression.
- OSSSample: OSS sample.
- ONSSample: ONS sample.
- ODPSSample: MaxCompute sample.
- MNSSample: MNS sample.
- LoghubSample: LogHub sample.

4.1.3.2. Dependencies

Test data (in the data directory)

- The_Sorrows_of_Young_Werther.txt: used as the input data for WordCount (MapReduce or Spark).
- patterns.txt: the filter characters for the WordCount (MapReduce) job.
- u.data: the data of the test table of the sample.hive script.
- abalone: the test data for linear regression algorithms.

JAR dependencies (in the lib directory)

tutorial.jar: the JAR dependencies required for the sample.pig job.

4.1.3.3. Preparations

This project provides some test data. You can directly upload the test data to OSS for use.

You can also prepare test data for other samples such as MaxCompute, MNS, ONS, and Log Service.

- (Optional) Create Log Service data. For more information, see the user guide for Log Service.
- (Optional) Create MaxCompute projects and tables. For more information, see the quick start for MaxCompute.
- (Optional) Create ONS data. For more information, see the quick start for ONS.
- (Optional) Create MNS data. For more information, see the console user guide for MNS.

4.1.3.4. Concepts

- OSS URI: specifies the input and output data sources for a job. It is similar to hdfs://. The format of OSS URI follows oss://accessKeyId:accessKeySecret@bucket.endpoint/a/b/c.txt.
- The combination of the AccessKey ID and AccessKey secret of your Alibaba Cloud account is the key for you to access an Alibaba Cloud API. You can obtain them from the administrator.

4.1.3.5. Cluster-based running

Spark

- SparkWordCount:

```
spark-submit --class SparkWordCount examples-1.0-SNAPSHOT-shaded.jar <inputPath> <outputPath>
    <numPartition>
```

The parameters are described as follows:

- inputPath: the data input path.
- outputPath: the data output path.
- numPartition: the number of RDD partitions of the input data.

- SparkPi:

```
spark-submit --class SparkPi examples-1.0-SNAPSHOT-shaded.jar
```

- OSSSample:

```
spark-submit --class OSSSample examples-1.0-SNAPSHOT-shaded.jar <inputPath>
    <numPartition>
```

The parameters are described as follows:

- inputPath: the data input path.
- numPartition: the number of RDD partitions of the input data.

- ONSSample:

```
spark-submit --class ONSSample examples-1.0-SNAPSHOT-shaded.jar <accessKeyId> <accessKeySecret>
    <consumerId> <topic> <subExpression> <parallelism>
```

The parameters are described as follows:

- accessKeyId: the AccessKey ID of your Alibaba Cloud cloud.
- accessKeySecret: the AccessKey secret corresponding to the AccessKey ID.
- consumerId: the identifier of a type of consumers. The consumers with the same consumer ID generally receive and consume a type of message, and use the same consumption logic.
- topic: Each message queue has a topic.
- subExpression: For more information, see Message filtering.
- parallelism: the number of receivers to consume messages in the queue.

- ODPSSample:

```
spark-submit --class ODPSSample examples-1.0-SNAPSHOT-shaded.jar <accessKeyId>
    <accessKeySecret> <envType> <project> <table> <numPartitions>
```

The parameters are described as follows:

- accessKeyId: the AccessKey ID of your Alibaba Cloud cloud.
- accessKeySecret: the AccessKey secret corresponding to the AccessKey ID.
- envType: the environment type. A value of 0 indicates the public network, and 1 indicates the internal network. Select 0 for local debugging, and select 1 for execution on E-MapReduce.
- project: the basic unit of operation in E-MapReduce. It is similar to the concept of Database or Schema in traditional databases, and sets the boundary for multi-user isolation and access control.

  - table: the database table.
  - numPartition: the number of RDD partitions of the input data.

- MNSSample:

```
spark-submit --class MNSSample examples-1.0-SNAPSHOT-shaded.jar <queueName> <accessKeyId>
    <accessKeySecret> <endpoint>
```

  The parameters are described as follows:

  - queueName: the name of the queue.
  - accessKeyId: the AccessKey ID of your Alibaba Cloud cloud.
  - accessKeySecret: the AccessKey secret corresponding to the AccessKey ID.
  - endpoint: the address for accessing the queue data.

- LoghubSample:

```
spark-submit --class LoghubSample examples-1.0-SNAPSHOT-shaded.jar <sls project> <sls logstore>
    logstore> <loghub group name> <sls endpoint> <access key id> <access key secret> <batch
   seconds>
```

  The parameters are described as follows:

  - sls project: the name of the Log Service project.
  - sls logstore: the name of the Logstore.
  - loghub group name: the name of the group that consumes Logstore data in jobs. You can specify a name as needed. For jobs with the same project name and Logstore name, they consume data in the Logstore collaboratively if they have the same group name. However, if the jobs have different group names, they consume data in the Logstore in an isolated manner.
  - sls endpoint: For more information, see Log Service endpoint.
  - accessKeyId: the AccessKey ID of your Alibaba Cloud cloud.
  - accessKeySecret: the AccessKey secret corresponding to the AccessKey ID.
  - batch interval seconds: the interval between Spark Streaming jobs. Unit: seconds.

- LinearRegression:

```
spark-submit --class LinearRegression examples-1.0-SNAPSHOT-shaded.jar <inputPath>
    <numPartitions>
```

  The parameters are described as follows:

  - inputPath: the data input path.
  - numPartition: the number of RDD partitions of the input data.

MapReduce

WordCount:

```
hadoop jar examples-1.0-SNAPSHOT-shaded.jar WordCount -Dwordcount.case.sensitive=true
    <inputPath> <outputPath> -skip <patternPath>
```

The parameters are described as follows:

- inputPath: the data input path.
- outputPath: the data output path.
- patternPath: the file that contains characters to be filtered. You can use data/patterns.txt.

Hive

```
hive -f sample.hive -hiveconf inputPath=<inputPath>
```

The parameters are described as follows:

inputPath: the data input path.

Pig

```
pig -x mapreduce -f sample.pig -param tutorial=<tutorialJarPath> -param input=<inputPath> -param result=
<resultPath>
```

The parameters are described as follows:

- tutorialJarPath: the JAR dependencies. You can use lib/tutorial.jar.
- inputPath: the data input path.
- resultPath: the data output path.

> Notice
>
> - If you run a program in E-MapReduce, upload the testing data and the JAR dependencies to OSS. The path rule must follow the definition of OSS URI, as described above.
> - If you run a program in a cluster, the testing data and the JAR dependencies can be stored locally.

4.1.3.6. Local running

This topic describes how to run the Spark program locally to access the data sources of Alibaba Cloud, such as OSS. If you want to debug and run Spark locally, we recommend that you use development tools such as Intellij IDEA or Eclipse. Pay attention to this especially when your machine runs Windows. Otherwise, you must configure the Hadoop and Spark runtime environments on the machine.

4.1.3.6.1. IntelliJ IDEA

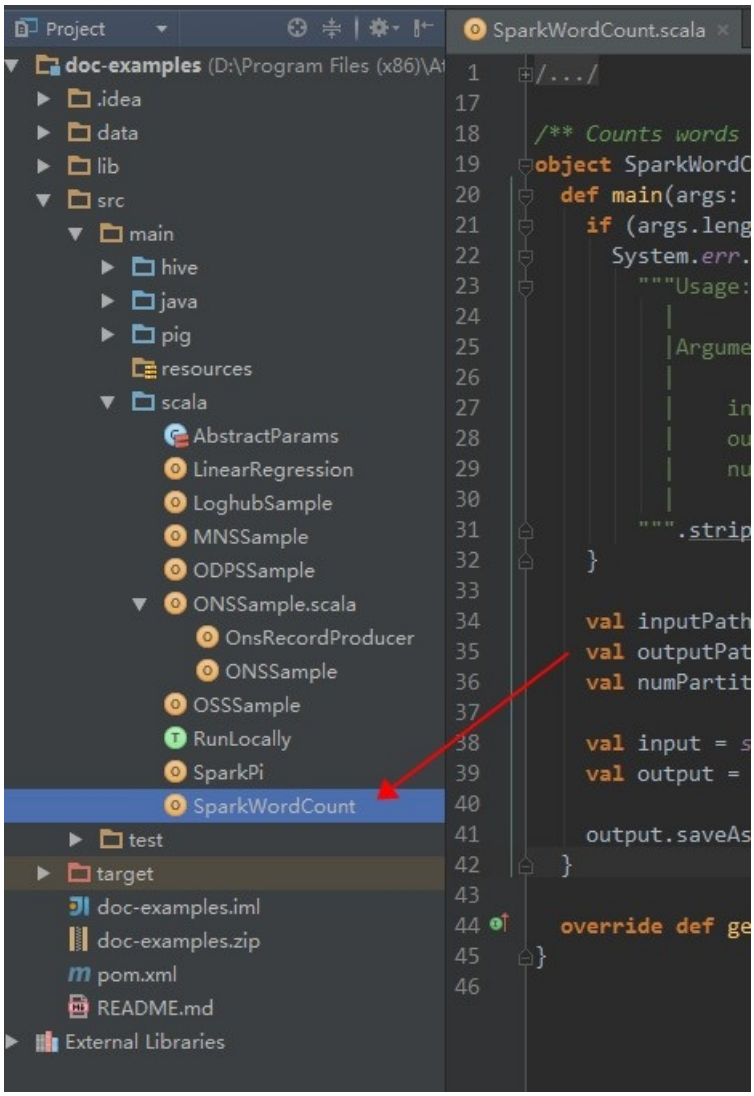
Preparations

Install IntelliJ IDEA, Maven, Maven plugin for IntelliJ IDEA, Scala, and Scala plugin for Intellij IDEA.

Development process

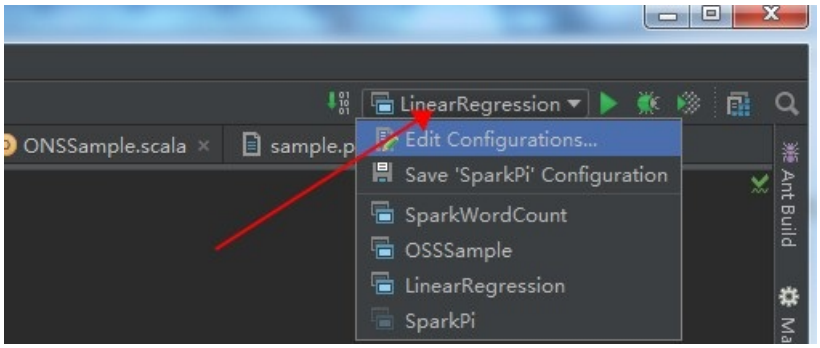
1. Double-click SparkWordCount to go to the SparkWordCount.scala page, as shown in SparkWordCount.scala page.

SparkWordCount.scala page



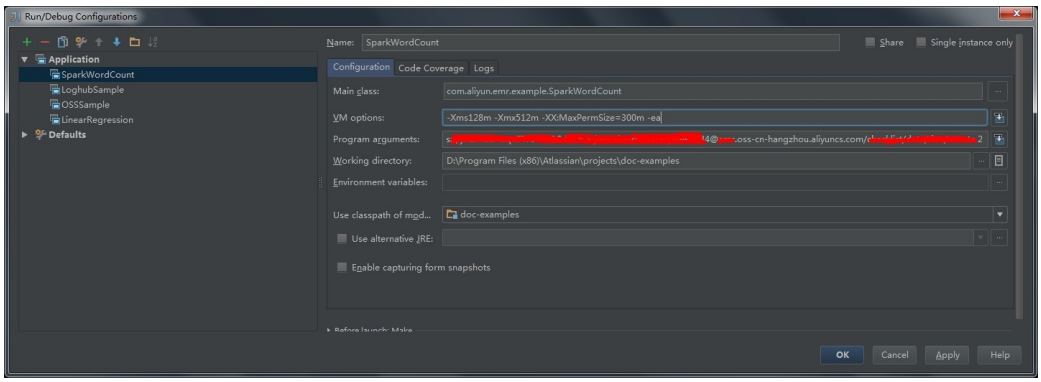
2. Go to the Run/Debug Configurations, page as shown in Run/Debug Configurations page.

Run/Debug Configurations page



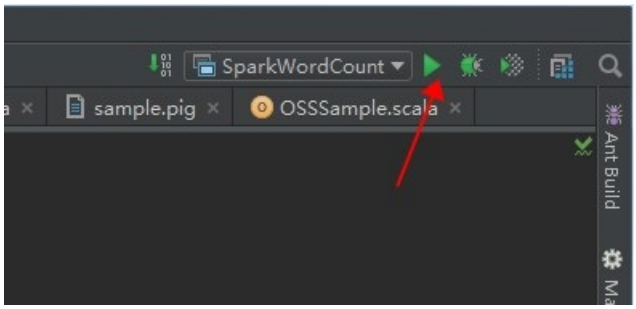
3. Select SparkWordCount in the left-side navigation pane, and enter the required job parameters in the job parameter box, as shown in Set the job parameters.

Set the job parameters



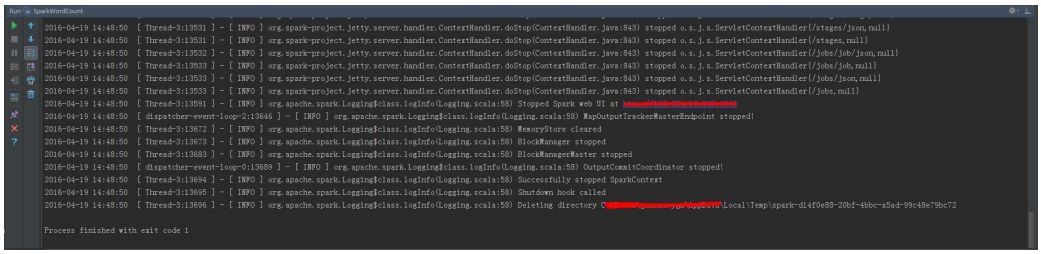
4. Click **OK**.
5. Click the **Run** icon to run the job, as shown in Run the job.

   Run the job



6. View the job execution logs, as shown in View job logs.

   View job logs



4.1.3.6.2. Scala IDE for Eclipse

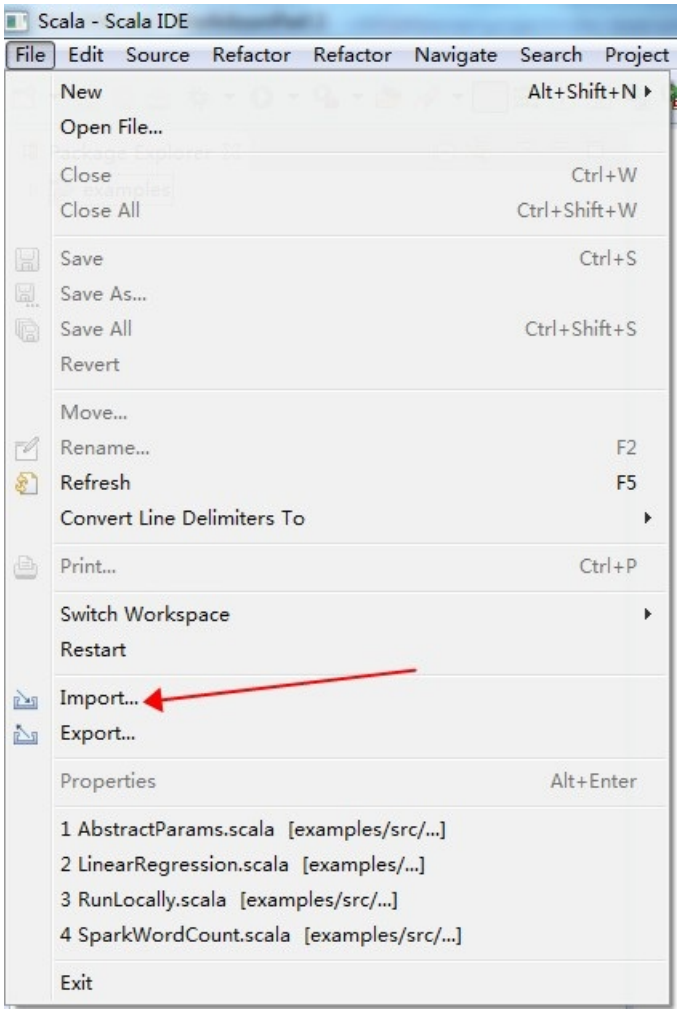
Preparations

Install the Scala IDE for Eclipse, Maven, and the Maven plugin for Eclipse.

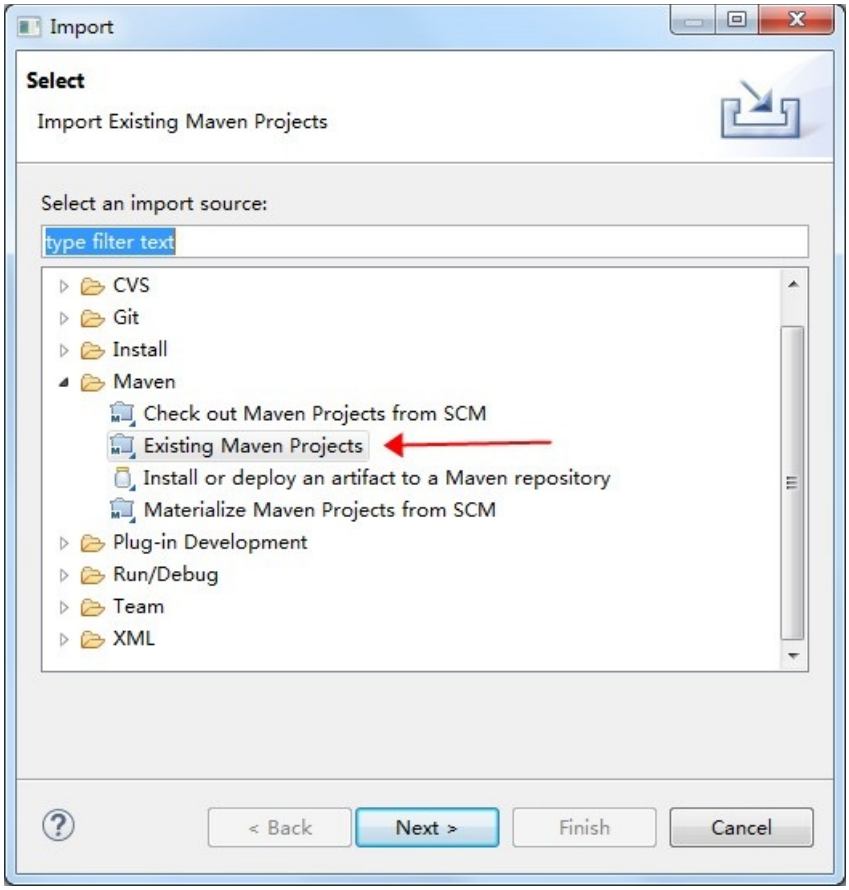
Development process

1. Import a project, as shown in Select Import, Select an import source, and Select Maven Projects.

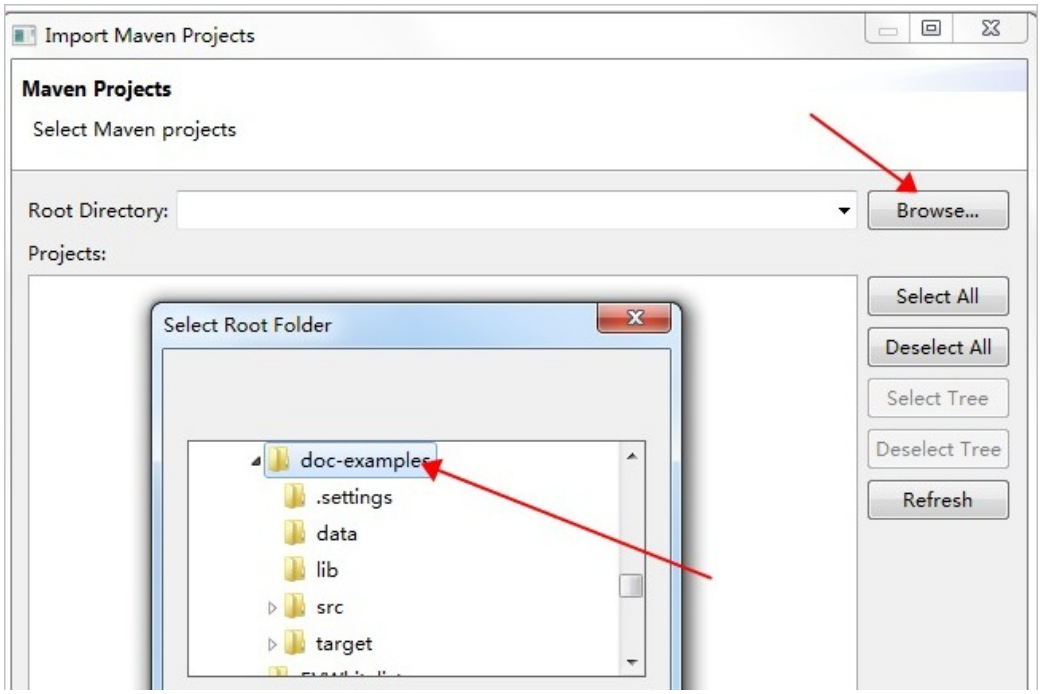
   Select Import



Select an import source



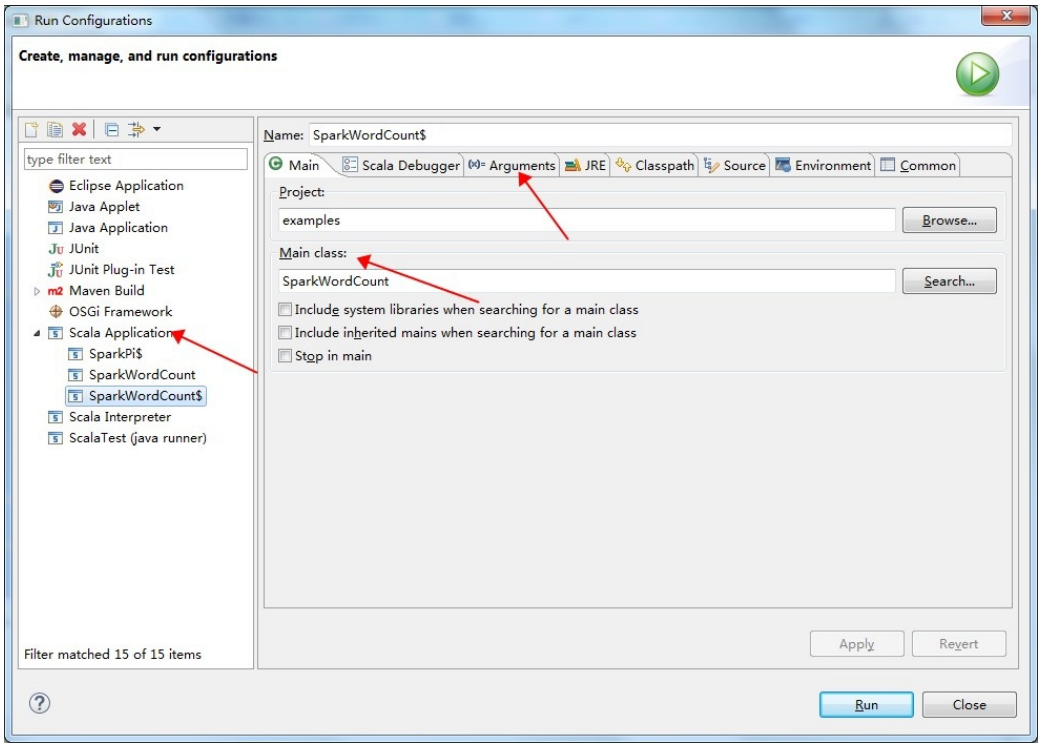
Select Maven Projects



2. Run as a Maven build. The shortcut is Alt+Shift+X, M. You can also right-click your project name and choose Run As > Maven build.
3. Right-click on the job to run after it is compiled, and select Run Configurations to go to the Run Configurations page.

4. On the Run Configurations page, select Scala Application and configure the Main Class and other parameters of the job, as shown in Run Configurations page.

   Run Configurations page



5. Click **Run**.
6. View the output logs in the console, as shown in View the output logs.

   View the output logs

4.2. Hadoop development

4.2.1. Parameter description

The following table describes the parameters used in Hadoop code.

Parameter description

Parameter	Default value	Description
fs.oss.accessKeyId	N/A	The AccessKey ID required for accessing OSS. This parameter is optional.
fs.oss.accessKeySecret	N/A	The AccessKey secret required for accessing OSS. This parameter is optional.
fs.oss.securityToken	N/A	The STS token required for accessing OSS. This parameter is optional.
fs.oss.endpoint	N/A	The endpoint used for accessing OSS. This parameter is optional.
fs.oss.multipart.thread.number	5	The number of concurrencies used by OSS for the upload part - copy operation.
fs.oss.copy.simple.max.byte	134217728	The file size limit for copying files between buckets in OSS through common APIs.
fs.oss.multipart.split.max.byte	67108864	The file multipart limit for copying files between buckets in OSS through common APIs.
fs.oss.multipart.split.number	5	The number of multipart files for copying files between buckets in OSS through common APIs. By default, the number is the same as the number of concurrencies used.
fs.oss.impl	com.aliyun.fs.oss.nat.NativeOssFileSystem	The implementation class for the native OSS file system.
fs.oss.buffer.dirs	/mnt/disk1,/mnt/disk2,...	The directory of the local temporary files in OSS. By default, the data disk in the cluster is used.
fs.oss.buffer.dirs.exists	false	Specifies whether the OSS temporary directory exists.
fs.oss.client.connection.timeout	50000	The timeout period for the connection to OSS client. Unit: milliseconds.
fs.oss.client.socket.timeout	50000	The timeout period for the connection to OSS client socket. Unit: milliseconds.

Parameter	Default value	Description
fs.oss.client.connection.ttl	-1	The connection alive time.
fs.oss.connection.max	1024	The maximum number of connections allowed.
io.compression.codec.snappy.native	false	Specifies whether the Snappy files are in the standard Snappy format. By default, Hadoop recognizes Snappy files edited in Hadoop.

4.2.2. Create and run MapReduce jobs

4.2.2.1. Process OSS data in MapReduce

To read data from and write data to OSS in MapReduce, you need to configure the following parameters:

```
conf.set("fs.oss.accessKeyId", "${accessKeyId}");
conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
conf.set("fs.oss.endpoint","${endpoint}");
```

The parameters are described as follows:

- **${accessKeyId}**: the AccessKey ID of your account.
- **${accessKeySecret}**: the AccessKey secret corresponding to the AccessKey ID.
- **${endpoint}**: the network used for accessing OSS. It depends on the region where your cluster resides. The corresponding OSS instance must be in the region where your cluster resides.

4.2.2.2. Word count

This topic describes how to read text from OSS and conduct a word count.

Procedure

1. Write a program. Take the Java code as an example. Modify the WordCount sample code on the Hadoop official website. Add the configuration of the AccessKey ID and AccessKey secret in the code so that the job has the permission to access OSS files.

```
package org.apache.hadoop.examples;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class EmrWordCount {
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();
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
if (otherArgs.length < 2) {
System.err.println("Usage: wordcount <in> [<in>...] <out>");
System.exit(2);
}
```

```
conf.set("fs.oss.accessKeyId", "${accessKeyId}");
    conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
    conf.set("fs.oss.endpoint","${endpoint}");
Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(EmrWordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
for (int i = 0; i < otherArgs.length - 1; ++i) {
FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
}
FileOutputFormat.setOutputPath(job,
new Path(otherArgs[otherArgs.length - 1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

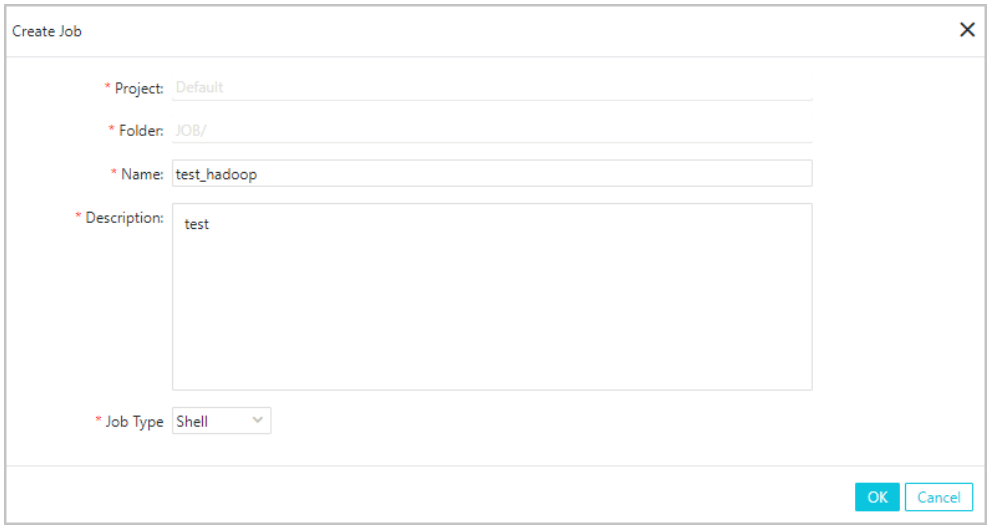
2. Compile the program. Configure the JDK and Hadoop environments and then run the following code:

```
mkdir wordcount_classes
javac -classpath ${HADOOP_HOME}/share/hadoop/common/hadoop-common-2.6.0.jar:${HADOOP_HO
ME}/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.6.0.jar:${HADOOP_HOME}/share/hado
op/common/lib/commons-cli-1.2.jar -d wordcount_classes EmrWordCount.java
jar cvf wordcount.jar -C wordcount_classes
```

3. Create a job.
   i. Upload the JAR file prepared in the previous step to OSS. You can switch to the OSS console to perform the operation. For example, assume that the path of the JAR file on OSS is oss://emr/jars/wordcount.jar and the input and output paths are oss://emr/data/WordCount/Input and oss://emr/data/WordCount/Output.

ii. Create a job in the EMR console, as shown in Create a job.

Create a job



4. Create and run a workflow. Create a workflow on the EMR console and add the created job to the workflow. Then, save and **run** the job.

4.2.2.3. Use Maven to manage MapReduce jobs

When your project grows larger in size, it becomes considerably complicated to manage. We recommend that you use Maven or similar software management tools to manage projects.

The procedure is as follows:

Procedure

1. Install Maven. First, make sure that you have installed Maven.
2. Generate the project framework. At the root directory of your project (assume that the root directory of your project is D:/workspace), run the following command:

   ```
   mvn archetype:generate -DgroupId=com.aliyun.emr.hadoop.examples -DartifactId=wordcountv2 -Darc
   hetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
   ```

   Maven automatically generates an empty sample project in the path D:/workspace/wordcountv2 (the same as the path specified for artifactId). The project contains a basic pom.xml file and an App class (the path of the class package is the same as that specified for groupId).
3. Add Hadoop dependencies. Open the project with an Integrated Development Environment (IDE) and edit the pom.xml file. Add the following content to the dependencies:

```
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-mapreduce-client-common</artifactId>
<version>2.6.0</version>
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-common</artifactId>
<version>2.6.0</version>
</dependency>
```

4. Write code. Add a new class named WordCount2.java at the same directory level as that of the App class in the com.aliyun.emr.hadoop.examples package. The code is as follows:

```
package com.aliyun.emr.hadoop.examples;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Counter;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.StringUtils;
public class WordCount2 {
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
static enum CountersEnum { INPUT_WORDS }
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
```

```
private boolean caseSensitive;
private Set<String> patternsToSkip = new HashSet<String>();
private Configuration conf;
private BufferedReader fis;
@Override
public void setup(Context context) throws IOException,
InterruptedException {
conf = context.getConfiguration();
        caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
if (conf.getBoolean("wordcount.skip.patterns", true)) {
            URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
for (URI patternsURI : patternsURIs) {
Path patternsPath = new Path(patternsURI.getPath());
String patternsFileName = patternsPath.getName().toString();
                parseSkipFile(patternsFileName);
}
}
}
private void parseSkipFile(String fileName) {
try {
            fis = new BufferedReader(new FileReader(fileName));
String pattern = null;
while ((pattern = fis.readLine()) ! = null) {
                patternsToSkip.add(pattern);
}
} catch (IOException ioe) {
System.err.println("Caught exception while parsing the cached file '"
+ StringUtils.stringifyException(ioe));
}
}
@Override
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
String line = (caseSensitive) ?
                value.toString() : value.toString().toLowerCase();
for (String pattern : patternsToSkip) {
            line = line.replaceAll(pattern, "");
}
StringTokenizer itr = new StringTokenizer(line);
while (itr.hasMoreTokens()) {
```

```
            word.set(itr.nextToken());
            context.write(word, one);
Counter counter = context.getCounter(CountersEnum.class.getName(),
CountersEnum.INPUT_WORDS.toString());
            counter.increment(1);
}
}
}
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();
       conf.set("fs.oss.accessKeyId", "${accessKeyId}");
      conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
       conf.set("fs.oss.endpoint","${endpoint}");
GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
String[] remainingArgs = optionParser.getRemainingArgs();
if (!( remainingArgs.length ! = 2 || remainingArgs.length ! = 4)) {
System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");
System.exit(2);
}
Job job = Job.getInstance(conf, "word count");
       job.setJarByClass(WordCount2.class);
       job.setMapperClass(TokenizerMapper.class);
       job.setCombinerClass(IntSumReducer.class);
       job.setReducerClass(IntSumReducer.class);
       job.setOutputKeyClass(Text.class);
       job.setOutputValueClass(IntWritable.class);
            job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
```

```
List<String> otherArgs = new ArrayList<String>();
for (int i=0; i < remainingArgs.length; ++i) {
if ("-skip".equals(remainingArgs[i])) {
        job.addCacheFile(new Path(EMapReduceOSSUtil.buildOSSCompleteUri(remainingArgs[++i], co
nf)).toUri());
} else {
        otherArgs.add(remainingArgs[i]);
}
}
FileInputFormat.addInputPath(job, new Path(EMapReduceOSSUtil.buildOSSCompleteUri(otherArgs.ge
t(0), conf)));
FileOutputFormat.setOutputPath(job, new Path(EMapReduceOSSUtil.buildOSSCompleteUri(otherArgs
.get(1), conf)));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

See the following EMapReduceOSSUtil class code, which is located in the same directory as WordCount2:

```
package com.aliyun.emr.hadoop.examples;
import org.apache.hadoop.conf.Configuration;
public class EMapReduceOSSUtil {
private static String SCHEMA = "oss://";
private static String AKSEP = ":";
private static String BKTSEP = "@";
private static String EPSEP = ".";
private static String HTTP_HEADER = "http://";
/**
    * complete OSS uri
    * convert uri like: oss://bucket/path to
        oss://accessKeyId:accessKeySecret@bucket.endpoint/path
    * ossref do not need this
    *
    * @param oriUri original OSS uri
    */
public static String buildOSSCompleteUri(String oriUri, String akId, String akSecret, String endpoint) {
if (akId == null) {
System.err.println("miss accessKeyId");
return oriUri;
}
if (akSecret == null) {
```

```
System.err.println("miss accessKeySecret");
return oriUri;
}
if (endpoint == null) {
System.err.println("miss endpoint");
return oriUri;
}
int index = oriUri.indexOf(SCHEMA);
if (index == -1 || index ! = 0) {
return oriUri;
}
int bucketIndex = index + SCHEMA.length();
int pathIndex = oriUri.indexOf("/", bucketIndex);
String bucket = null;
if (pathIndex == -1) {
        bucket = oriUri.substring(bucketIndex);
} else {
        bucket = oriUri.substring(bucketIndex, pathIndex);
}
StringBuilder retUri = new StringBuilder();
     retUri.append(SCHEMA)
.append(akId)
.append(AKSEP)
.append(akSecret)
.append(BKTSEP)
.append(bucket)
.append(EPSEP)
.append(stripHttp(endpoint));
if (pathIndex > 0) {
        retUri.append(oriUri.substring(pathIndex));
}
return retUri.toString();
}
public static String buildOSSCompleteUri(String oriUri, Configuration conf) {
return buildOSSCompleteUri(oriUri, conf.get("fs.oss.accessKeyId"), conf.get("fs.oss.accessKeySecret"),
conf.get("fs.oss.endpoint"));
}
private static String stripHttp(String endpoint) {
if (endpoint.startsWith(HTTP_HEADER)) {
return endpoint.substring(HTTP_HEADER.length());
```

```
}
return endpoint;
}
}
```

5. Compile, package, and upload the code. In the project directory, run the following command:

   ```
   mvn clean package -DskipTests
   ```

   You can see wordcountv2-1.0-SNAPSHOT.jar in the target directory of your project directory, which is the JAR package of the job. Upload the JAR package to OSS.

6. Create a job. Run the following command to create a job in E-MapReduce:

   ```
   jar ossref://yourBucket/yourPath/wordcountv2-1.0-SNAPSHOT.jar com.aliyun.emr.hadoop.examples.WordCount2 -Dwordcount.case.sensitive=true oss://yourBucket/yourPath/The_Sorrows_of_Young_Werther.txt oss://yourBucket/yourPath/output -skip oss://yourBucket/yourPath/patterns.txt
   ```

   In the command, yourBucket is one of your OSS buckets, and yourPath is a path in this bucket. You need to replace yourPath with where the Hive script is saved. You need to download the `oss://yourBucket/yourPath/The_Sorrows_of_Young_Werther.txt` and `oss://yourBucket/yourPath/patterns.txt` files for processing related resources, and then store them in OSS. You can download the resources needed for the jobs from the following links and store the resources in the corresponding directory of OSS.

   Download: The_Sorrows_of_Young_Werther.txt and patterns.txt.

7. Create and run an execution plan. Create the execution plan in E-MapReduce, associate it with the job, and then run the execution plan.

4.2.3. Create and run a Hive job

4.2.3.1. Use OSS in Hive

To read and write OSS in Hive, you must add the AccessKey ID, AccessKey secret, and endpoint when using the OSS URI. The following example shows how to create an external table:

```
CREATE EXTERNAL TABLE eusers (
 userid INT)
LOCATION 'oss://emr/users';
```

To ensure successful access to OSS, you need to modify the OSS URI as follows:

```
CREATE EXTERNAL TABLE eusers (
 userid INT)
LOCATION 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/users';
```

The parameters are described as follows:

- **${accessKeyId}**: the AccessKey ID of your Alibaba Cloud account.

- **${accessKeySecret}**: the AccessKey secret corresponding to the AccessKey ID.
- **${endpoint}**: the network used for accessing OSS. It depends on the region where your cluster resides. The corresponding OSS instance must be in the region where your cluster resides.

4.2.3.2. Use Tez as the computing engine

Introduced in EMR V2.1.0, Tez is a computing framework that optimizes complex DAG scheduling jobs. It improves the operation speed of Hive jobs in many scenarios.

You can optimize your jobs by using Tez as the execution engine:

```
set hive.execution.engine=tez
```

Example 1

Perform the following steps:

1. Write the following script, save it as hiveSample1.sql, and then upload it to OSS.

```
USE DEFAULT;
set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
set hive.stats.autogather=false;
 DROP TABLE emrusers;
 CREATE EXTERNAL TABLE emrusers (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '\t'
 STORED AS TEXTFILE
 LOCATION 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/yourpath';
 SELECT COUNT(*) FROM emrusers;
 SELECT * from emrusers limit 100;
 SELECT movieid,count(userid) as usercount from emrusers group by movieid order by usercount desc l
imit 50;
```

2. Prepare test data. Click the following link to download the required resources for Hive jobs and store the resources to the directory in your OSS instance.

   Download resources: public test data

3. Create a job. Run the following command to create a job in EMR:

```
-f ossref://${bucket}/yourpath/hiveSample1.sql
```

   In this example, ${bucket} indicates one of your OSS buckets, and yourpath indicates a path in the bucket. You need to replace yourpath with the path where the Hive script is stored.

4. Create and run a workflow. Create a workflow. You can associate it with an existing cluster or create a cluster as needed. Then, associate the workflow with the created job. Save and run the job.

Example 2

Take scan in HiBench as an example. If the input and output are both from OSS, you need to perform the following operations to run the Hive job. Modify the code based on the settings of the AccessKey ID, AccessKey secret, and storage path. Set the OSS path in the following format:

```
oss://${accesskeyId}:${accessKeySecret}@${bucket}.${endpoint}/object/path
```

Perform the following steps:

1. Write the following script:

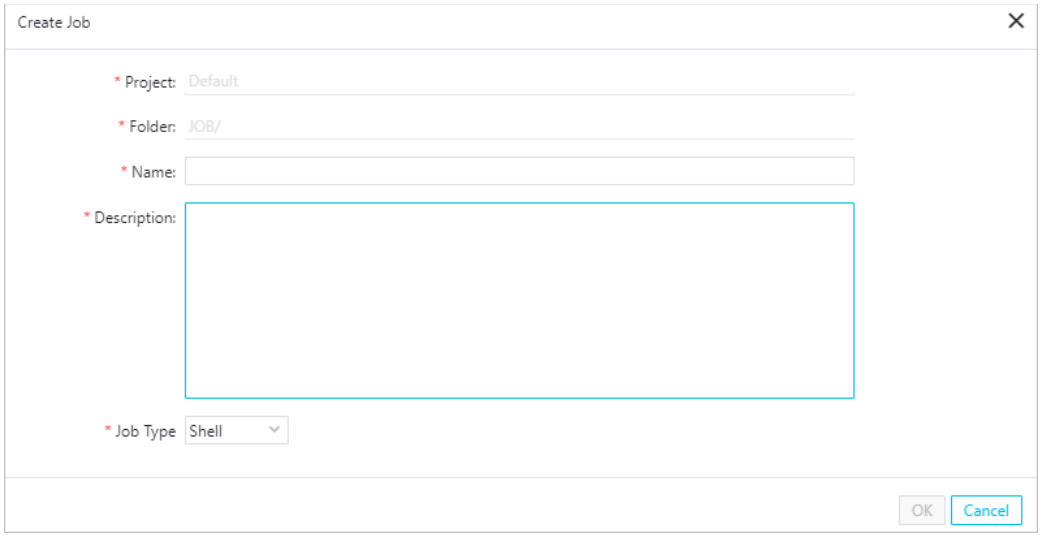
```
USE DEFAULT;
set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
set mapreduce.job.maps=12;
set mapreduce.job.reduces=6;
set hive.stats.autogather=false;
DROP TABLE uservisits;
CREATE EXTERNAL TABLE uservisits (sourceIP STRING,destURL STRING,visitDate STRING,adRevenue DO
UBLE,userAgent STRING,countryCode STRING,languageCode STRING,searchWord STRING,duration INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS SEQUENCEFILE LOCATION 'oss://${Acce
ssKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/sample-data/hive/Scan/Input/uservisits';
```

2. Prepare test data. Click the following link to download the required resources for Hive jobs and store the resources to the directory in your OSS instance.

   Download resources: uservisits

3. Create a job. Store the script created in step 1 in the storage path in OSS, such as `oss://emr/jars/scan.hive`. Then, create a job, as shown in Create a job.

   Create a job



4. Create and run a workflow. Create a workflow. You can associate it with an existing cluster or create a cluster as needed. Then, associate the workflow with the created job. Save and run the job.

4.2.4. Create and run a Pig job

Use OSS in Pig

Use the following format for OSS path:

```
oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/${path}
```

The parameters are described as follows:

- **${accessKeyId}**: the AccessKey ID of your Alibaba Cloud account.
- **${accessKeySecret}**: the AccessKey secret corresponding to the AccessKey ID.
- **${bucket}**: the bucket corresponding to the AccessKey ID.
- **${endpoint}**: the network used for accessing OSS. It depends on the region where your cluster resides. The corresponding OSS instance must be in the region where your cluster resides.
- **${path}**: the path of the file in the bucket.

Take the script1-hadoop.pig file in Pig as an example. Upload tutorial. jar and excite.log.bz2 in Pig to OSS. Assume that the upload paths are respectively `oss://emr/jars/tutorial.jar` and `oss://emr/data/excite.log.bz2`.

Follow these steps:

1. Write a script Modify the path of the JAR file and the input and output paths in the script, as shown below. Set the OSS path in the following format:

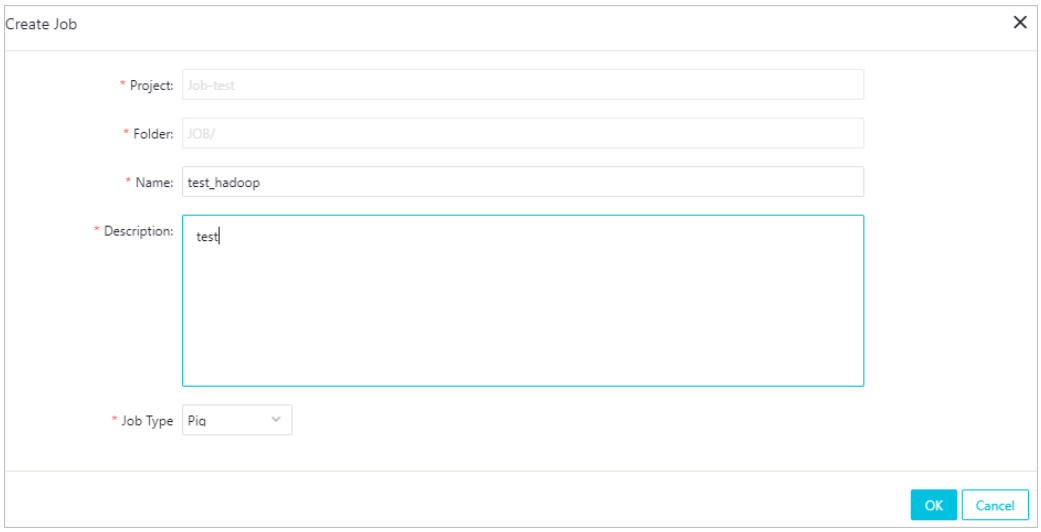
```
oss://${accesskeyId}:${accessKeySecret}@${bucket}.${endpoint}/object/path
/*
 * Licensed to the Apache Software Foundation (ASF) under one
* or more contributor license agreements.  See the NOTICE file
* distributed with this work for additional information
```

```
 distributed with this work for additional information
* regarding copyright ownership.  The ASF licenses this file
* to you under the Apache License, Version 2.0 (the
* "License"); you may not use this file except in compliance
* with the License.  You may obtain a copy of the License at
*
*    http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
     software
* distributed under the License is distributed on an "AS IS"
     BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied.
* See the License for the specific language governing permissions
     and
* limitations under the License.
*/
-- Query Phrase Popularity (Hadoop cluster)
-- This script processes a search query log file from the Excite search engine and finds search phrases th
at occur with particular high frequency during certain times of the day.
-- Register the tutorial JAR file so that the included UDFs can be called in the script.
REGISTER oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/data/tutorial.jar;
-- Use the PigStorage function to load the excite log file into the ▒raw▒ bag as an array of records.
-- Input: (user,time,query)
raw = LOAD 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/data/excite.log.bz2' USIN
G PigStorage('\t') AS (user, time, query);
-- Call the NonURLDetector UDF to remove records if the query field is empty or a URL.
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
-- Call the ToLower UDF to change the query field to lowercase.
clean2 = FOREACH clean1 GENERATE user, time,    org.apache.pig.tutorial.ToLower(query) as query;
-- Because the log file only contains queries for a single day, we are only interested in the hour.
-- The excite query log timestamp format is YYMMDDHHMMSS.
-- Call the ExtractHour UDF to extract the hour (HH) from the time field.
houred = FOREACH clean2 GENERATE user, org.apache.pig.tutorial.ExtractHour(time) as hour, query;
-- Call the NGramGenerator UDF to compose the n-grams of the query.
ngramed1 = FOREACH houred GENERATE user, hour, flatten(org.apache.pig.tutorial.NGramGenerator(q
uery)) as ngram;
-- Use the DISTINCT command to get the unique n-grams for all records.
ngramed2 = DISTINCT ngramed1;
-- Use the GROUP command to group records by n-gram and hour.
```

```
hour_frequency1 = GROUP ngramed2 BY (ngram, hour);
-- Use the COUNT function to get the count (occurrences) of each n-gram.
hour_frequency2 = FOREACH hour_frequency1 GENERATE flatten($0), COUNT($1) as count;
-- Use the GROUP command to group records by n-gram only.
-- Each group now corresponds to a distinct n-gram and has the count for each hour.
uniq_frequency1 = GROUP hour_frequency2 BY group::ngram;
-- For each group, identify the hour in which this n-gram is used with a particularly high frequency.
-- Call the ScoreGenerator UDF to calculate a "popularity" score for the n-gram.
uniq_frequency2 = FOREACH uniq_frequency1 GENERATE flatten($0), flatten(org.apache.pig.tutorial.Sc
oreGenerator($1));
-- Use the FOREACH-GENERATE command to assign names to the fields.
uniq_frequency3 = FOREACH uniq_frequency2 GENERATE $1 as hour, $0 as ngram, $2 as score, $3 as cou
nt, $4 as mean;
-- Use the FILTER command to move all records with a score less than or equal to 2.0.
filtered_uniq_frequency = FILTER uniq_frequency3 BY score > 2.0;
-- Use the ORDER command to sort the remaining records by hour and score.
ordered_uniq_frequency = ORDER filtered_uniq_frequency BY hour, score;
-- Use the PigStorage function to store the results.
-- Output: (hour, n-gram, score, count, average_counts_among_all_hours)
STORE ordered_uniq_frequency INTO 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint
}/data/script1-hadoop-results' USING PigStorage();
```

2. Create a job. Store the script written in Step 1 to OSS. Assume that the storage path is `oss://emr/jars/script1-hadoop.pig`. Create a job in E-MapReduce, as shown in Create a job.

   Create a job

3. Create and run an execution plan. Create an execution plan in E-MapReduce and add the created Pig job to the execution plan. Select Manually executed as the scheduling policy and save the job. Return to the previous page and click Run now to run the script1-hadoop job.

4.2.5. Create a Hadoop streaming job

Use Python to create a Hadoop streaming job

The mapper code is as follows:

```
#! /usr/bin/env python
import sys
for line in sys.stdin:
  line = line.strip()
  words = line.split()
for word in words:
print '%s\t%s' % (word, 1)
```

The reducer code is as follows:

```
#! /usr/bin/env python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
  line = line.strip()
  word, count = line.split('\t', 1)
try:
    count = int(count)
except ValueError:
continue
if current_word == word:
    current_count += count
else:
if current_word:
print '%s\t%s' % (current_word, current_count)
    current_count = count
    current_word = word
if current_word == word:
print '%s\t%s' % (current_word, current_count)
```

Assume that the mapper code is saved as the /home/hadoop/mapper.py file and the reducer code is saved as the /home/hadoop/reducer.py file. The input and output paths are /tmp/input and /tmp/output in HDFS, respectively. You can run the following Hadoop command in the E-MapReduce cluster to create and run a Hadoop streaming job:

```
hadoop jar /usr/lib/hadoop-current/share/hadoop/tools/lib/hadoop-streaming-*.jar -file
    /home/hadoop/mapper.py -mapper mapper.py -file /home/hadoop/reducer.py -reducer reducer.py
    -input /tmp/hosts -output /tmp/output
```

4.2.6. Process Table Store data in Hive

4.2.6.1. Connect Table Store to Hive

Prepare a data table

Create a table named pet and set the name field as the primary key.

Data table pet

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	02/04/1993	-
Claws	Gwen	cat	m	03/17/1994	-
Buffy	Harold	dog	f	05/13/1989	-
Fang	Benny	dog	m	08/27/1990	-
Bowser	Diane	dog	m	08/31/1979	07/29/1995
Chirpy	Gwen	bird	f	09/11/1998	-
Whistler	Gwen	bird	-	12/09/1997	-
Slim	Benny	snake	m	04/29/1996	-
Puffball	Diane	hamster	f	03/30/1999	-

How to process Table Store data in Hive

Example:

1. Run the following command:

   ```
   $ HADOOP_HOME=YourHadoopDir HADOOP_CLASSPATH=emr-tablestore-<version>.jar:tablestore-4.1.
   0-jar-with-dependencies.jar:joda-time-2.9.4.jar bin/hive
   ```

2. Create an external table.

```
CREATE EXTERNAL TABLE pet
(name STRING, owner STRING, species STRING, sex STRING, birth STRING, death STRING)
 STORED BY 'com.aliyun.openservices.tablestore.hive.TableStoreStorageHandler'
 WITH SERDEPROPERTIES(
"tablestore.columns.mapping"="name,owner,species,sex,birth,death")
 TBLPROPERTIES (
"tablestore.endpoint"="YourEndpoint",
"tablestore.access_key_id"="YourAccessKeyId",
"tablestore.access_key_secret"="YourAccessKeySecret",
"tablestore.table.name"="pet");
```

3. Insert data to the external table.

```
INSERT INTO pet VALUES("Fluffy", "Harold", "cat", "f", "1993-02-04", null);
INSERT INTO pet VALUES("Claws", "Gwen", "cat", "m", "1994-03-17", null);
INSERT INTO pet VALUES("Buffy", "Harold", "dog", "f", "1989-05-13", null);
INSERT INTO pet VALUES("Fang", "Benny", "dog", "m", "1990-08-27", null);
INSERT INTO pet VALUES("Bowser", "Diane", "dog", "m", "1979-08-31", "1995-07-29");
INSERT INTO pet VALUES("Chirpy", "Gwen", "bird", "f", "1998-09-11", null);
INSERT INTO pet VALUES("Whistler", "Gwen", "bird", null, "1997-12-09", null);
INSERT INTO pet VALUES("Slim", "Benny", "snake", "m", "1996-04-29", null);
INSERT INTO pet VALUES("Puffball", "Diane", "hamster", "f", "1999-03-30", null);
```

4. Query data.

```
> SELECT * FROM pet;
Bowser Diane  dog   m    1979-08-31 1995-07-29
Buffy Harold dog   f    1989-05-13    NULL
Chirpy Gwen   bird  f    1998-09-11    NULL
Claws Gwen   cat   m    1994-03-17    NULL
Fang Benny  dog   m    1990-08-27    NULL
Fluffy Harold cat   f    1993-02-04    NULL
Puffball Diane  hamster f    1999-03-30    NULL
Slim Benny  snake  m    1996-04-29    NULL
Whistler Gwen   bird   NULL   1997-12-09    NULL
> SELECT * FROM pet WHERE birth > "1995-01-01";
Chirpy Gwen   bird  f    1998-09-11    NULL
Puffball Diane  hamster f    1999-03-30    NULL
Slim Benny  snake  m    1996-04-29    NULL
Whistler Gwen   bird   NULL   1997-12-09    NULL
```

4.2.6.2. Data type conversion

Data type conversion

	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
INTEGER	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported	Supported (with loss of precision)	Supported (with loss of precision)			
DOUBLE	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported			
BOOLEAN							Supported		
STRING								Supported	
BINARY									Supported

4.2.6.3. Appendix

For the complete sample code, see: Process Table Store data in Hive.

4.2.7. Process Table Store data in MapReduce

4.2.7.1. Connect Table Store to MapReduce

Prepare a data table

Create a table named pet and set the name field as the primary key.

Data table pet

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	02/04/1993	-
Claws	Gwen	cat	m	03/17/1994	-
Buffy	Harold	dog	f	05/13/1989	-
Fang	Benny	dog	m	08/27/1990	-
Bowser	Diane	dog	m	08/31/1979	07/29/1995

name	owner	species	sex	birth	death
Chirpy	Gwen	bird	f	09/11/1998	-
Whistler	Gwen	bird	-	12/09/1997	-
Slim	Benny	snake	m	04/29/1996	-
Puffball	Diane	hamster	f	03/30/1999	-

How to consume Table Store data in MapReduce

Example:

```
public class RowCounter {
public static class RowCounterMapper
extends Mapper<PrimaryKeyWritable, RowWritable, Text, LongWritable> {
private final static Text agg = new Text("TOTAL");
private final static LongWritable one = new LongWritable(1);
@Override public void map(PrimaryKeyWritable key, RowWritable value,
Context context) throws IOException, InterruptedException {
        context.write(agg, one);
}
}
public static class IntSumReducer
extends Reducer<Text,LongWritable,Text,LongWritable> {
@Override public void reduce(Text key, Iterable<LongWritable> values,
Context context) throws IOException, InterruptedException {
long sum = 0;
for (LongWritable val : values) {
          sum += val.get();
}
        context.write(key, new LongWritable(sum));
}
}
private static RangeRowQueryCriteria fetchCriteria() {
RangeRowQueryCriteria res = new RangeRowQueryCriteria("pet");
    res.setMaxVersions(1);
List<PrimaryKeyColumn> lower = new ArrayList<PrimaryKeyColumn>();
List<PrimaryKeyColumn> upper = new ArrayList<PrimaryKeyColumn>();
    lower.add(new PrimaryKeyColumn("name", PrimaryKeyValue.INF_MIN));
    upper.add(new PrimaryKeyColumn("name", PrimaryKeyValue.INF_MAX));
    res.setInclusiveStartPrimaryKey(new PrimaryKey(lower));
    res.setExclusiveEndPrimaryKey(new PrimaryKey(upper));
```

```
return res;
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
    job.setJarByClass(RowCounter.class);
    job.setMapperClass(RowCounterMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
    job.setInputFormatClass(TableStoreInputFormat.class);
TableStore.setCredential(job, accessKeyId, accessKeySecret, securityToken);
TableStore.setEndpoint(job, endpoint, instance);
TableStoreInputFormat.addCriteria(job, fetchCriteria());
FileOutputFormat.setOutputPath(job, new Path(outputPath));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

How to write data to Table Store in MapReduce

Example:

```
public static class OwnerMapper
extends Mapper<PrimaryKeyWritable, RowWritable, Text, MapWritable> {
@Override public void map(PrimaryKeyWritable key, RowWritable row,
Context context) throws IOException, InterruptedException {
PrimaryKeyColumn pet = key.getPrimaryKey().getPrimaryKeyColumn("name");
Column owner = row.getRow().getLatestColumn("owner");
Column species = row.getRow().getLatestColumn("species");
MapWritable m = new MapWritable();
    m.put(new Text(pet.getValue().asString()),
new Text(species.getValue().asString()));
    context.write(new Text(owner.getValue().asString()), m);
}
}
public static class IntoTableReducer
extends Reducer<Text,MapWritable,Text,BatchWriteWritable> {
@Override public void reduce(Text owner, Iterable<MapWritable> pets,
Context context) throws IOException, InterruptedException {
List<PrimaryKeyColumn> pkeyCols = new ArrayList<PrimaryKeyColumn>();
    pkeyCols.add(new PrimaryKeyColumn("owner"
```

```
    pkeyCols.add(new PrimaryKeyColumn("owner",
PrimaryKeyValue.fromString(owner.toString())));
PrimaryKey pkey = new PrimaryKey(pkeyCols);
List<Column> attrs = new ArrayList<Column>();
for(MapWritable petMap: pets) {
for(Map.Entry<Writable, Writable> pet: petMap.entrySet()) {
Text name = (Text) pet.getKey();
Text species = (Text) pet.getValue();
        attrs.add(new Column(name.toString(),
ColumnValue.fromString(species.toString())));
}
}
RowPutChange putRow = new RowPutChange(outputTable, pkey)
.addColumns(attrs);
BatchWriteWritable batch = new BatchWriteWritable();
    batch.addRowChange(putRow);
    context.write(owner, batch);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, TableStoreOutputFormatExample.class.getName());
  job.setMapperClass(OwnerMapper.class);
  job.setReducerClass(IntoTableReducer.class);
  job.setMapOutputKeyClass(Text.class);
  job.setMapOutputValueClass(MapWritable.class);
  job.setInputFormatClass(TableStoreInputFormat.class);
  job.setOutputFormatClass(TableStoreOutputFormat.class);
TableStore.setCredential(job, accessKeyId, accessKeySecret, securityToken);
TableStore.setEndpoint(job, endpoint, instance);
TableStoreInputFormat.addCriteria(job, ...) ;
TableStoreOutputFormat.setOutputTable(job, outputTable);
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

4.2.7.2. Appendix

For the complete sample code, see:

- Read data from Table Store in MapReduce
- Write data to Table Store in MapReduce

4.3. Create an HBase cluster and use the HBase storage service

4.3.1. Create an HBase cluster and use the HBase storage service

To make better use of HBase, we recommend that you use the following configurations when creating a cluster:

- Enable access from the public network.
- Select the zone where the server that accesses HBase is located. Do not randomly assign a zone.
- Select four or more hardware nodes, including the master and slave nodes. E-MapReduce creates NameNode, DataNode, JournalNode, HMaster, RegionServer, and ZooKeeper roles on these nodes.
- Select quad-core, 16 GB memory or 8-core, 32 GB memory for servers. Lower configurations may impair the stability of HBase clusters.
- Select SSD for the data disk for cost-effectiveness. You can select basic disk for the data disk for services that are not frequently accessed but require a large amount of storage space.
- Configure the data capacity as needed.
- HBase cluster supports memory expansion.

4.3.2. HBase configuration

When creating an HBase cluster, you can use the software configuration feature on the page to optimize and modify the default parameters of HBase in specific scenarios. See the following example:

```
{
 "configurations": [
  {
   "classification": "hbase-site",
   "properties": {
    "hbase.hregion.memstore.flush.size": "268435456",
    "hbase.regionserver.global.memstore.size": "0.5",
    "hbase.regionserver.global.memstore.lowerLimit": "0.6"
   }
  }
 ]
}
```

Default configurations for HBase cluster describes some default configurations for HBase clusters.

Default configurations for HBase cluster

Key	Value
zookeeper.session.timeout	180000
hbase.regionserver.global.memstore.size	0.35
hbase.regionserver.global.memstore.lowerLimit	0.3
hbase.hregion.memstore.flush.size	128 MB

4.3.3. Access HBase

> **Note**
>
> - To improve the network performance, we recommend that you initiate an access request to HBase cluster created by E-MapReduce from the ECS instance in the same zone.
> - The ECS instance accessing HBase cluster must be in the same security group as the HBase cluster. Otherwise, the access fails. Therefore, if you create a Hadoop, Spark, or Hive cluster in E-MapReduce, and the cluster you create needs to access HBase, make sure that you select the same security group as the HBase cluster.

After you have created an HBase cluster in the E-MapReduce console, you can use the HBase storage service. The procedure is as follows:

1. Obtain the IP address of the master node and the ZooKeeper access address. On the Cluster Overview page in the E-MapReduce console, you can view the IP addresses of the master nodes and ZooKeeper access address (internal IP address).

   For more information about master nodes with public IP addresses enabled, you can browse the HMaster WEB UI (localhost:16010).

2. Use HBase Shell to connect to the master node of a cluster over SSH. You can connect to the master node of a cluster over SSH, switch to the HDFS user, and then use HBase Shell to access the cluster. For more information about HBase Shell, see Apache HBase official website.

   ```
   [root@emr-header-1 ~]# su hdfs
   [hadoop@emr-header-1 root]$ hbase shell
   HBase Shell; enter 'help<RETURN>' for list of supported commands.
   Type "exit<RETURN>" to leave the HBase Shell
   Version 1.1.1, r374488, Fri Aug 21 09:18:22 CST 2015
   hbase(main):001:0>
   ```

3. Use HBase Shell to access the cluster from another ECS node (within the same security group). Download the HBase V1.x resource package from the Apache HBase official website (Download link), decompress the resource package, modify conf/hbase-site.xml, and then add the ZooKeeper address of the cluster, as shown in the following section:

```
<configuration>
<property>
<name>hbase.zookeeper.quorum</name>
<value>$ZK_IP1,$ZK_IP2,$ZK_IP3</value>
</property>
</configuration>
```

Then you can run the bin/hbase shell command to access the cluster.

If the ECS instance is created through E-MapReduce, you only need to modify the following file:

```
hbase-site.xml
```

That is, you do not need to download the HBase V1.x resource package.

The configuration file for HBase V3.2.0 or later is as follows:

```
/etc/ecm/hbase-conf/hbase-site.xml
```

The configuration file for HBase versions earlier than V3.2.0 is as follows:

```
/etc/emr/hbase-conf/hbase-site.xml
```

4. To access the HBase cluster through APIs, add the following Maven dependency:

```
<groupId>org.apache.hbase</groupId>
<artifactId>hbase-client</artifactId>
<version>1.1.1</version>
```

For more information about the Apache HBase development, see Apache HBase official website.

4.3.4. Example

Prerequisites

The ECS instance accessing the HBase cluster must be in the same security group as the HBase cluster.

Allow Spark to access HBase

For more information, see spark-hbase-connector.

Allow Hadoop to access HBase

For more information, see HBase MapReduce Examples.

Allow Hive to access HBase

Only Hive running in clusters of E-MapReduce V1.2.0 or later can access the HBase cluster. Follow these steps:

1. Log on to the Hive cluster, and modify hosts by adding the following line:

```
$zk_ip emr-cluster //$zk_i p is the IP address of the ZooKeeper node in the HBase cluster.
```

2. For more information about how to operate Hive, see Hive HBase Integration.

4.3.5. HBase backup

The HBase cluster created in E-MapReduce allows you to use the snapshot feature integrated into HBase to back up HBase tables, and export the backup data to Alibaba Cloud Object Storage Service (OSS).

Example:

1. Create an HBase cluster.
2. Create a table.

```
>create 'test','cf'
```

3. Add data.

```
> put 'test','a','cf:c1',1
> put 'test','a','cf:c2',2
> put 'test','b','cf:c1',3
> put 'test','b','cf:c2',4
> put 'test','c','cf:c1',5
> put 'test','c','cf:c2',6
```

4. Create a snapshot.

```
hbase snapshot create -n test_snapshot -t test
```

View the snapshot. The following section shows the details of the snapshot.

```
>list_snapshots
SNAPSHOT          TABLE + CREATION TIME
 test_snapshot       test (Sun Sep 04 20:31:00 +0800 2016)
1 row(s) in 0.2080 seconds
```

5. Export the snapshot to OSS.

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot test_snapshot -copy-to oss://$a
ccessKeyId:$accessKeySecret@$bucket.oss-cn-hangzhou-internal.aliyuncs.com/hbase/snapshot/test
```

> Note Access OSS through an internal network endpoint.

6. Create another HBase cluster.
7. Export the snapshot from OSS.

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot test_snapshot -copy-from oss://
$accessKeyId:$accessKeySecret@$bucket.oss-cn-hangzhou-internal.aliyuncs.com/hbase/snapshot/tes
t
      -copy-to /hbase/
```

8. Restore data from the snapshot.

```
>restore _snapshot 'test_snapshot'
```

```
>scan 'test'
ROW            COLUMN+CELL
 a             column=cf:c1, timestamp=1472992081375, value=1
 a             column=cf:c2, timestamp=1472992090434, value=2
 b             column=cf:c1, timestamp=1472992104339, value=3
 b             column=cf:c2, timestamp=1472992099611, value=4
 c             column=cf:c1, timestamp=1472992112657, value=5
 c             column=cf:c2, timestamp=1472992118964, value=6
3 row(s) in 0.0540 seconds
```

9. Create a table based on the snapshot.

```
>clone_snapshot 'test_snapshot','test_2'
```

```
>scan 'test_2'
ROW            COLUMN+CELL
 a             column=cf:c1, timestamp=1472992081375, value=1
 a             column=cf:c2, timestamp=1472992090434, value=2
 b             column=cf:c1, timestamp=1472992104339, value=3
 b             column=cf:c2, timestamp=1472992099611, value=4
 c             column=cf:c1, timestamp=1472992112657, value=5
 c             column=cf:c2, timestamp=1472992118964, value=6
3 row(s) in 0.0540 seconds
```

4.4. Data transmission application - Sqoop

4.4.1. Install Sqoop

Sqoop is an open-source application that is used to transfer data between different data storage software. It supports various data storage software.

> **Notice** Sqoop has been integrated with E-MapReduce since E-MapReduce V1.3. If you are using E-MapReduce V1.3 or later, you can skip this section.

If your E-MapReduce version is earlier than V1.3, Sqoop is not integrated with it. If you need to use Sqoop, you can manually install it by performing the following operations:

1. Download Sqoop V1.4.6 from the official website (Click to download). If you cannot open the `sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz file,` download the file from the mirror site `http://mirror.bit.edu.cn/apache/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz` by running the following command:

   ```
   wget http://mirror.bit.edu.cn/apache/sqoop/1.4.6/sqoop-1.4.6.bin__ha doop-2.0.4-alpha.tar.gz
   ```

2. Run the following command to decompress the `sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz` file to the master node:

   ```
   tar zxf sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
   ```

3. To export data from MySQL, install the MySQL driver. Download the latest version on the official website (Click to download), or run the following command to download the latest version (take version 5.1.38 as an example):

   ```
   wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.38.tar.gz
   ```

4. Extract the JAR file to the lib folder in the Sqoop folder.

4.4.2. Data transmission

Common scenarios:

- From MySQL to HDFS
- From HDFS to MySQL
- From MySQL to Hive
- From Hive to MySQL
- From MySQL to OSS
- From OSS to MySQL
- Use SQL to import data

> **Notice**
>
> You must switch your user account to hadoop before running the following command:
>
> ```
> su hadoop
> ```

4.4.2.1. Import data from MySQL to HDFS

You can run the following command on the master node of a cluster:

```
sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <username> --password <password> --table <tablename> --target-dir <hdfs-dir>
```

The parameters are described as follows:

- dburi: the access link of a database, such as jdbc:mysql://192.168.1.124:3306/. If any parameter is included in the access link, enclose the access link within single quotation marks ('), such as 'jdbc:mysql://192.168.1.124:3306/mydatabase? useUnicode=true'.
- dbname: the name of a database, such as user.
- username: the username that is used to log on to a database.
- password: the password that is associated with the username.
- tablename: the name of a MySQL table.
- hdfs-dir: the HDFS directory that you import data into, such as /user/hive/result.

For more information about parameters, see Sqoop Import.

4.4.2.2. Import data from HDFS to MySQL

You need to create a MySQL table with a data structure matching that in OSS, and then specify the path of the data file to be exported on the master node of the cluster.

```
sqoop export --connect jdbc:mysql://<dburi>/<dbname> --username <username> --password <password> --t
able <tablename> --export-dir
    <hdfs-dir>
```

The parameters are described as follows:

- dburi: the access link of a database, such as jdbc:mysql://192.168.1.124:3306/. If any parameter is included in the access link, enclose the access link within single quotation marks ('), such as 'jdbc:mysql://192.168.1.124:3306/mydatabase? useUnicode=true'.
- dbname: the name of a database, such as user.
- username: the username that is used to log on to a database.
- password: the password that is associated with the username.
- tablename: the name of a MySQL table.
- hdfs-dir: the HDFS directory from which you import data into MySQL, such as /user/hive/result.

For more information about parameters, see Sqoop export.

4.4.2.3. Import data from MySQL to Hive

A Hive table is created while you import data to Hive. Run the following command:

```
sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <username> --password <password> --
table <tablename>
    --fields-terminated-by "\t" --lines-terminated-by "\n" --hive-import --target-dir
    <hdfs-dir> --hive-table <hive-tablename>
```

The parameters are described as follows:

- dburi: the access link of a database, such as jdbc:mysql://192.168.1.124:3306/. If any parameter is included in the access link, enclose the access link within single quotation marks ('), such as jdbc:mysql://192.168.1.124:3306/mydatabase? useUnicode=true.

- dbname: the name of a database, such as user.
- username: the username that is used to log on to a database.
- password: the password that is associated with the username.
- tablename: the name of a MySQL table.
- hdfs-dir: the HDFS directory from which you import data into MySQL, such as /user/hive/result.
- hive-tablename: the name of a table that is used to import data to Hive, such as xxx.yyy.

For more information about parameters, see Sqoop Import.

4.4.2.4. Import data from Hive to MySQL

You only need to specify the HDFS path corresponding to the Hive table. For more information, see the command in Import data from HDFS to MySQL.

4.4.2.5. Import data from MySQL to OSS

The process is similar to importing data from MySQL into HDFS, except for the configuration of the target-dir parameter. You can run the following command on the master node of a cluster:

> **Notice**
>
> - The endpoint of an OSS host can be an internal network endpoint, public network endpoint, or VPC endpoint. For a classic network, you must specify an internal network endpoint. For example, the OSS internal network endpoint of the China (Hangzhou) region is oss-cn-hangzhou-**internal**.aliyuncs.com. For a VPC, you must specify a VPC endpoint. For example, the OSS VPC endpoint of the China (Hangzhou) region is **vpc100**-oss-cn-hangzhou.aliyuncs.com.
> - Currently, when you import data into OSS, you cannot specify the delete-target-dir parameter. Otherwise, the error message Wrong FS occurs. If you want to overwrite the data in the previous OSS directory, you can run the `hadoop fs -rm -r osspath` command to remove this OSS directory before using Sqoop.

```
sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <username> --password <password> --
table <tablename> --target-dir
    <oss-dir>
```

The parameters are described as follows:

- dburi: the access link of a database, such as jdbc:mysql://192.168.1.124:3306/. If any parameter is included in the access link, enclose the access link within single quotation marks ('), such as jdbc:mysql://192.168.1.124:3306/mydatabase? useUnicode=true.
- dbname: the name of a database, such as user.
- username: the username that is used to log on to a database.
- password: the password that is associated with the username.
- tablename: the name of a MySQL table.
- oss-dir: the OSS directory that you import data into, such as `oss://<accessid>:<accesskey>@<bucketname>.oss-cn-hangzhou-internal.aliyuncs.com/result`.

For more information about parameters, see Sqoop import.

4.4.2.6. Import data from OSS to MySQL

The process is similar to importing data from MySQL to HDFS, except for the configuration of the export-dir parameter. You need to create a MySQL table with a data structure matching that in OSS, and then specify the path of the data file to be exported on the master node of the cluster.

```
sqoop export --connect jdbc:mysql://<dburi>/<dbname> --username <username> --password <password> --table <tablename> --export-dir <oss-dir>
```

The parameters are described as follows:

- dburi: the access link of a database, such as jdbc:mysql://192.168.1.124:3306/. If any parameter is included in the access link, enclose the access link within single quotation marks ('), such as 'jdbc:mysql://192.168.1.124:3306/mydatabase? useUnicode=true'.
- dbname: the name of a database, such as user.
- username: the username that is used to log on to a database.
- password: the password that is associated with the username.
- tablename: the name of a MySQL table.
- oss-dir: the OSS directory that you write data into, such as `oss://<accessid>:<accesskey>@<bucketname>.oss-cn-hangzhou-internal.aliyuncs.com/result`.

> **Notice** The endpoint of an OSS host can be an internal network endpoint, public network endpoint, or VPC endpoint. For a classic network, you must specify an internal network endpoint. For example, the OSS internal network endpoint of the China (Hangzhou) region is oss-cn-hangzhou-**internal**.aliyuncs.com. For a VPC, you must specify a VPC endpoint. For example, the OSS VPC endpoint of the China (Hangzhou) region is **vpc100**-oss-cn-hangzhou.aliyuncs.com.

For more information about parameters, see Sqoop export.

4.4.2.7. Use SQL to import data

In addition to importing a set of MySQL tables, you can also use SQL to import data as follows:

```
sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <username> --password <password> --query <query-sql> --split-by <sp-column> --hive-import --hive-table <hive-tablename> --target-dir <hdfs-dir>
```

Parameter description:

- dburi: the access link of a database, such as jdbc:mysql://192.168.1.124:3306/. If any parameter is included in the access link, enclose the access link within single quotation marks ('), such as 'jdbc:mysql://192.168.1.124:3306/mydatabase? useUnicode=true'.
- dbname: the name of a database, such as user.
- username: the username that is used to log on to a database.
- password: the password that is associated with the username.
- query-sql: the query statement, such as "SELECT * FROM profile WHERE id>1 AND \$CONDITIONS". You

must enclose the query statement that ends with AND \$CONDITIONS within double quotation marks (").

- sp-column: specifies the name of a column to be split. In general, the value of this parameter is the primary key of the MySQL table.
- hdfs-dir: the HDFS directory from which you import data into MySQL, such as /user/hive/result.
- hive-tablename: the name of a table that is used to import data to Hive, such as xxx.yyy.

For more information about available parameters, see Sqoop Query Import.

5.DataHub

5.1. SDK overview

This topic describes the SDKs provided by DataHub.

Package	Description
aliyun-sdk-datahub	The SDK is used to perform basic operations in DataHub. The SDK package provides API operations on basic objects in DataHub, including projects, topics, and shards.

DataHub provides SDKs for Java, Python, and Go. The following topics describe how to use DataHub SDKs for Java and Python.

5.2. Prerequisites

Before using the SDK for Java, you must perform the following operations:

- Create an account with authorization to access MaxCompute and obtain the AccessKey pair for the account. For more information about how to obtain and view your AccessKey pair in the Apsara Stack console, see Obtain an AccessKey pair.
- Obtain the endpoint. For more information, see Obtain an endpoint.

5.2.1. Overview

Before using SDKs for MaxCompute, you must perform the following operations:

- Create an account that is authorized to access MaxCompute and obtain an AccessKey pair (AccessKey ID and AccessKey secret).
- Obtain the service endpoint.

5.2.2. Obtain an AccessKey pair

You can obtain an AccessKey pair in the Apsara Stack Cloud Management (ASCM) console.

5.2.2.1. 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.

5.2.2.2. Obtain the AccessKey pair of an organization

This topic describes how to obtain an AccessKey pair.

Prerequisites

Only the operation administrators or level-1 organization administrators can obtain the AccessKey pair of an organization.

Procedure

1. Log on to the Apsara Stack Cloud Management (ASCM) console as an administrator.
2. In the top navigation bar, click **Enterprise**.
3. In the left-side navigation pane of the **Enterprise** page, click **Organizations**.
4. In the Organizations navigation tree, click the icon on the right of an organization.
5. Select **AccessKey** from the shortcut menu.
6. In the message that appears, view the AccessKey pair of the organization.

   > **Note** An AccessKey pair is automatically allocated to a level-1 organization. Subordinate organizations of the level-1 organization use the AccessKey pair of the level-1 organization.

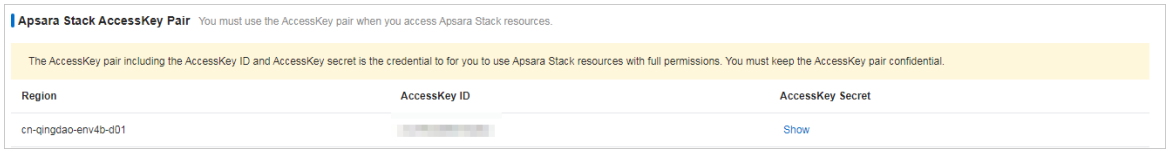
5.2.2.3. View the AccessKey pair of your Apsara Stack tenant account

To secure cloud resources, the system must verify the identity of visitors and ensure that they have the relevant permissions. You must obtain the AccessKey ID and AccessKey secret of your personal account to access cloud resources.

Procedure

1. Log on to the Apsara Stack Cloud Management (ASCM) console as an administrator.
2. In the upper-right corner of the homepage, move the pointer over the profile picture and click **User Information**.
3. In the **Apsara Stack AccessKey Pair** section of the User Information page, view your AccessKey

pair.



> **Note** The AccessKey pair is made up of the AccessKey ID and AccessKey secret. These credentials provide you full permissions on Apsara Stack resources. You must keep the AccessKey pair confidential.

5.2.3. Obtain the endpoint

This topic describes how to obtain the service endpoint of DataHub in the Apsara Stack Operations (ASO) console.

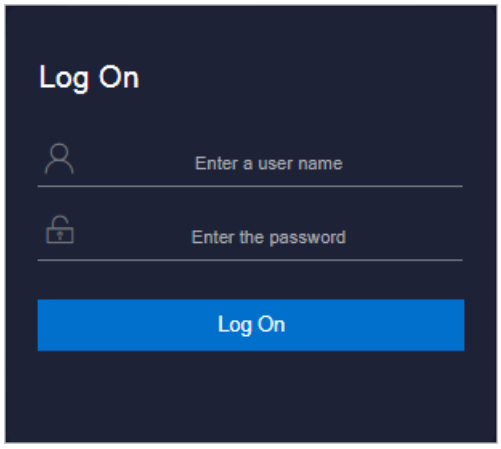
Context

- The URL, username, and password that are required to log on to the ASO console are obtained from deployment engineers or administrators.

  The URL of the ASO console is in the following format: *region-id*.aso.*intranet-domain-id*.com.
- A browser is available. We recommend that you use Google Chrome.

Procedure

1. Open your browser.
2. In the address bar, enter the URL *region-id*.aso.*intranet-domain-id*.com and press Enter.



> **Note** You can select a language from the drop-down list in the upper-right corner of the page.

3. Enter your username and password.

> **Note** You can obtain the username and password for logging on to the ASO console from deployment engineers or administrators.

   - By default, the following user roles are available:

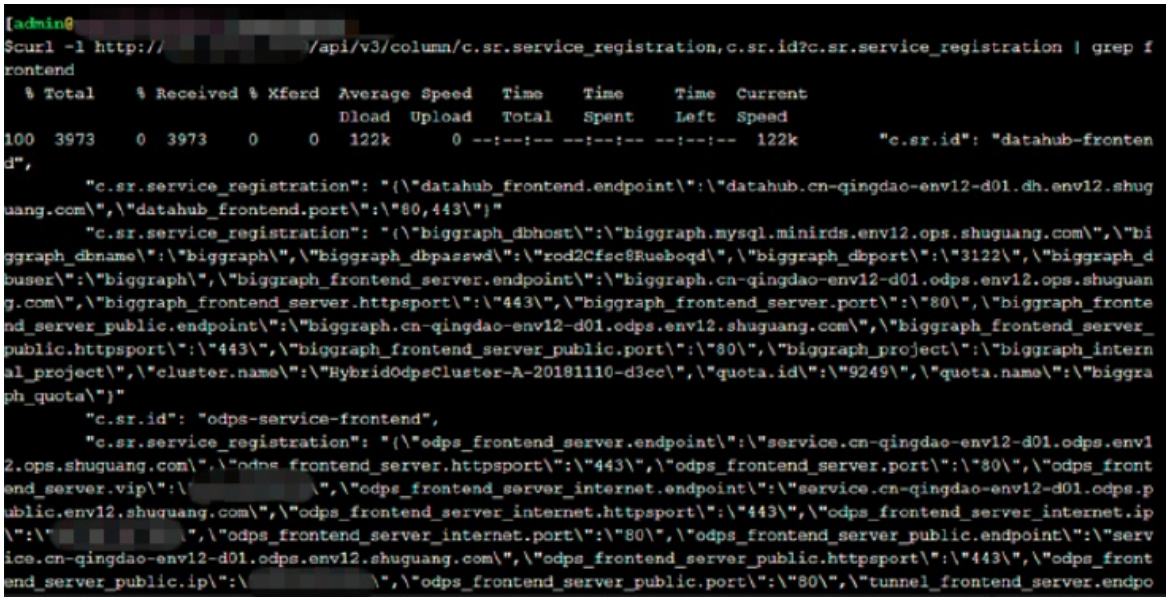
- Security administrator: the user who has the permissions to manage other users or roles.
- Auditor: the user who has the permissions to view audit logs.
- Sysadmin: the user who has more permissions than a security administrator and an auditor.

- When you log on to the ASO console for the first time, you must change the password as prompted.

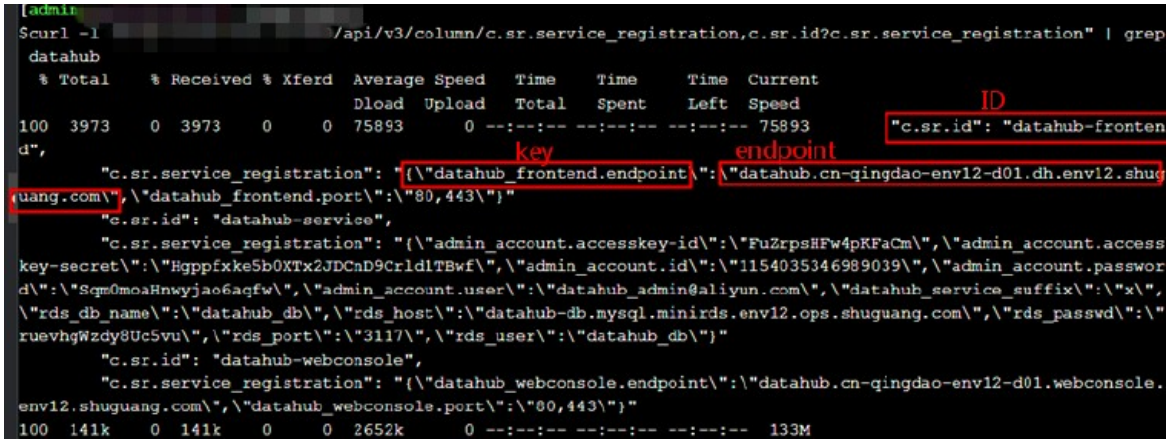
  To enhance security, make sure that the password meet the following requirements:

  - The password contains uppercase and lowercase letters.
  - The password contains digits.
  - The password contains special characters such as exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%).
  - The password is 10 to 20 characters in length.

4. Click **Log On** to go to the **ASO** console.
5. In the left-side navigation pane, choose **Products > Product List**. In the **Basic O&M** section of the **Apsara Stack O&M** pane, click **Apsara Infrastructure Management Framework**.
6. In the left-side navigation pane, choose **Operations > Machine Operations**. The **Machine Operations** page appears.
7. In the Machines search box, search for a host. Click **Terminal** in the Actions column of the host to log on to the host.
8. Run the following command on the host to obtain JSON data: **curl -l "http://127.0.0.1:7070/api/v3/column/c.sr.service_registration,c.sr.id?c.sr.service_registration"**.



9. Find the endpoint of *DataHub* based on the service ID (*datahub-frontend*) and the key value (*datahub_frontend.endpoint*) of *DataHub*.



5.3. DataHub SDK for Java

5.3.1. Overview

You do not need to download the DataHub SDK for Java. You only need to configure the SDK as a Maven dependency in a pom.xml file. Add the following dependency configuration to the pom.xml file:

```
<dependency>
  <groupId>com.aliyun.datahub</groupId>
  <artifactId>aliyun-sdk-datahub</artifactId>
  <version>2.9.2-public</version>
</dependency>
```

Note If you use DataHub V3.8.1 or earlier, use the preceding dependency configuration, where the SDK version is 2.9.2.

```
<dependency>
  <groupId>com.aliyun.datahub</groupId>
  <artifactId>aliyun-sdk-datahub</artifactId>
  <version>2.13.1-public</version>
</dependency>
```

Note If you use DataHub V3.8.1 or later, use the preceding dependency configuration, where the SDK version is 2.13.1.

5.3.2. Preparations

To access DataHub, you must have an account that is authorized to access DataHub, the AccessKey for the account, and an endpoint of DataHub. If you are using Apsara Stack V2, obtain the endpoint from the CMDB system. If you are using Apsara Stack V3, obtain the endpoint from the Apsara Infrastructure Management Framework console.

The sample code is shown as follows:

```
String accessId = "Your AccessId";
String accessKey = "Your AccessKey";
String endpoint = "http://XXXXXX";
// Specify the endpoint you have obtained.
AliyunAccount account = new AliyunAccount(accessId, accessKey);
DatahubConfiguration conf = new DatahubConfiguration(account, endpoint);
```

The following example shows how to initialize a DataHub client:

```
DatahubClient client = new DatahubClient(conf);
```

> **Note** After the DataHub client is initialized, you can call the DatahubClient operation to access DataHub. You can use this client to perform all operations on DataHub because the client is thread-safe.

5.3.3. Ingest tuple data

5.3.3.1. Create a tuple topic

The sample code is shown as follows:

```
RecordSchema schema = new RecordSchema();
schema.addField(new Field("a", FieldType.STRING));
schema.addField(new Field("b", FieldType.BIGINT));
int shardCount = 5;
int lifeCycle = 3;
String topicName = "topic_example";
String topicDesc = "topic_example_desc";
client.createTopic(projectName, topicName, shardCount, lifeCycle, RecordType.TUPLE, schema, topicDesc);
// Wait until the shards are in Active status.
client.waitForShardReady(projectName, topicName);
```

> **Note** In the sample code, lifeCycle indicates the time-to-live of each record in the topic.

5.3.3.2. Obtain the list of shards

The sample code is shown as follows:

```
ListShardResult listShardResult = client.listShard(projectName, topicName);
```

5.3.3.3. Write data into DataHub

The sample code is shown as follows:

```
List<RecordEntry> recordEntries = new ArrayList<RecordEntry>();
// Write into DataHub starting from any active shard returned by listShardResult. In this example, start from the first active shard.
String shardId = listShardResult.getShards().get(0).getShardId();
RecordEntry entry = new RecordEntry(schema);
entry.setString(0, "Test");
entry.setBigint(1, 5L);
entry.setShardId(shardId);
recordEntries.add(entry);
PutRecordsResult result = client.putRecords(projectName, topicName, recordEntries);
if (result.getFailedRecordCount() ! = 0) {
  List<ErrorEntry> errors = result.getFailedRecordError();
  // Process records failed to be written into DataHub.
  for (ErrorEntry e : result.getFailedRecordError()) {
    if (! e.getErrorcode().equals( “MalformedRecord” )) { // Retry the operation except that the MalformedRecord, NoSuchProject, and NoSuchTopic errors occur.}
}
```

Note A request body size cannot exceed 4 MB. For more information, see Data ingestion methods.

5.3.3.4. Consume data

You can use this SDK to obtain the cursor that points to the first record to be consumed. The cursor type can be **OLDEST**, **LATEST**, or **SYSTEM_TIME**. OLDEST: the cursor that points to the earliest record in DataHub. LATEST: the cursor that points to the latest record in DataHub. SYSTEM_TIME: the cursor that points to the first record ingested after the specified time.

The sample code is shown as follows:

```
GetCursorResult cursorRs = client.getCursor(projectName, topicName, shardId, GetCursorRequest.CursorType.OLDEST);
// If you want to obtain the cursor that points to the earliest record ingested during the last 24 hours, set CursorType to SYSTEM_TIME and set SYSTEM_TIME to System.currentTimeMillis() - 24 * 3600 * 1000 /* ms */.
int limit = 100;
String cursor = cursorRs.getCursor();
while (true) {
  try {
    GetBlobRecordsResult recordRs = client.getBlobRecords(projectName, topicName, shardId, cursor, limit);
    List<BlobRecordEntry> recordEntries = recordRs.getRecords();
    if (recordEntries.size() == 0) {
      // Data has not been updated. Try again later.
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    // The cursor that points to the next record to be consumed is obtained.
    cursor = recordRs.getNextCursor();
  } catch (InvalidCursorException ex) {
    // The cursor is invalid or has expired. Specify another record as the beginning of consumption.
    cursorRs = client.getCursor(projectName, topicName, shardId, GetCursorRequest.CursorType.OLDEST);
    cursor = cursorRs.getCursor();
  } catch (DatahubClientException ex) {
    // An error occurred while obtaining the cursor. Try again.
    System.out.printf(ex.getMessage());
    ex.printStackTrace();
  }
}
```

5.3.4. Ingest blob data

5.3.4.1. Create a blob topic

The sample code is shown as follows:

```
int shardCount = 5;
int lifeCycle = 3;
String topicName = "topic_example";
String topicDesc = "topic_example_desc";
client.createTopic(projectName, topicName, shardCount, lifeCycle, RecordType.BLOB, topicDesc);
// Wait until the shards are in Active status.
client.waitForShardReady(projectName, topicName);
```

5.3.4.2. Obtain the list of shards

The sample code is shown as follows:

```
ListShardResult listShardResult = client.listShard(projectName, topicName);
```

5.3.4.3. Write data into DataHub

The sample code is shown as follows:

```
List<BlobRecordEntry> recordEntries = new ArrayList<BlobRecordEntry>();
// Write into DataHub starting from any active shard returned by listShardResult. In this example, start from the first active shard.
String shardId = listShardResult.getShards().get(0).getShardId();
String data = String.valueOf(System.currentTimeMillis());
BlobRecordEntry entry = new BlobRecordEntry();
entry.setData(data.getBytes());
entry.setShardId(shardId);
recordEntries.add(entry);
PutBlobRecordsResult result = client.putBlobRecords(projectName, topicName, recordEntries);
if (result.getFailedRecordCount() ! = 0) {
  List<ErrorEntry> errors = result.getFailedRecordError();
  // Retry the operation except that the MalformedRecord, NoSuchProject, and NoSuchTopic errors occur.
}
```

> **Note** A request body size cannot exceed 4 MB.

5.3.4.4. Consume data

The sample code is shown as follows:

```
GetCursorResult cursorRs = client.getCursor(projectName, topicName, shardId, GetCursorRequest.CursorTy
pe.OLDEST);
// If you want to obtain the cursor that points to the earliest record ingested during the last 24 hours, set Cu
rsorType to SYSTEM_TIME and set SYSTEM_TIME to System.currentTimeMillis() - 24 * 3600 * 1000 /* ms */.
int size = 100;
String endCursor = cursorRes.GetCursor();
while (true) {
  try {
    GetBlobRecordsResult recordRs = client.getBlobRecords(projectName, topicName, shardId, cursor, limit)
;
    List<BlobRecordEntry> recordEntries = recordRs.getRecords();
    if(records.size() == 0) {
      // Data has not been updated. Try again later.
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    // The cursor that points to the next record to be consumed is obtained.
    cursor = recordRs.getNextCursor();
  } catch (InvalidCursorException ex) {
    // The cursor is invalid or has expired. Specify another record as the beginning of consumption.
    cursorRs = client.getCursor(projectName, topicName, shardId, GetCursorRequest.CursorType.OLDEST);
    cursor = cursorRs.getCursor();
  } catch (DatahubClientException ex) {
    // An error occurred while obtaining the cursor. Try again.
    System.out.println(response.getMessage());
    e.printStackTrace();
  }
}
```

> **Note** Before being consumed, blob records that have been base64 encoded during ingestion must be decoded.

5.3.5. Data ingestion methods

5.3.5.1. Ingest data based on shard IDs

By using this method, you can write data into a specific shard and ensure that data is written into each shard in a specified order.

The sample code is shown as follows:

```
// Create a client.
Account account = new AliyunAccount("your access id", "your access key");
DatahubConfiguration conf = new DatahubConfiguration(account, "datahub endpoint");
DatahubClient client = new DatahubClient(conf);
// Specify records to be ingested by DataHub.
RecordSchema schema = client.getTopic("projectName", "topicName").getRecordSchema();
List<RecordEntry> recordEntries = new ArrayList<~>();
RecordEntry entry = new RecordEntry(schema);
for (int i=0; i<entry.getFieldCount(); i++) {
  entry.setBigint(i, 1);
}
// Write into DataHub starting from any active shard returned by listShardResult. In this example, start from the first active shard.
String shardId = listShardResult.getShards().get(0).getShardId();
entry.setShardId(shardId);
recordEntries.add(entry);
// Write data into DataHub.
PutRecordsResult result = client.putRecords("projectName", "topicName", recordEntries);
```

5.3.5.2. Ingest data based on hash values

Specify a 128-bit hash value produced by MD5 message-digest algorithm. Use the hash value to map associated records to a shard based on the hash key range of the shards.

You cannot specify the ingestion order of records by using this method.

The sample code is shown as follows:

```
// Create a client.
Account account = new AliyunAccount("your access id", "your access key");
DatahubConfiguration conf = new DatahubConfiguration(account, "datahub endpoint");
DatahubClient client = new DatahubClient(conf);
// Specify records to be ingested by DataHub.
RecordSchema schema = client.getTopic("projectName", "topicName").getRecordSchema();
List<RecordEntry> recordEntries = new ArrayList<~>();
RecordEntry entry = new RecordEntry(schema);
for (int i=0; i<entry.getFieldCount(); i++) {
  entry.setBigint(i, 1);
}
entry.setHashKey("7FFFFFFFFFFFFFFD7FFFFFFFFFFFFFFD");
recordEntries.add(entry);
// Write the data into DataHub.
PutRecordsResult result = client.putRecords("projectName", "topicName", recordEntries);
```

5.3.5.3. Ingest data based on partition keys

Specify a partition key as a string. DataHub creates an MD5 hash from the string and maps the records to shards based on the hash key range of the shards.

You do not need to provide hash values by using this method.

The sample code is shown as follows:

```
// Create a client.
Account account = new AliyunAccount("your access id", "your access key");
DatahubConfiguration conf = new DatahubConfiguration(account, "datahub endpoint");
DatahubClient client = new DatahubClient(conf);
// Specify records to be ingested by DataHub.
RecordSchema schema = client.getTopic("projectName", "topicName").getRecordSchema();
List<RecordEntry> recordEntries = new ArrayList<~>();
RecordEntry entry = new RecordEntry(schema);
for (int i=0; i<entry.getFieldCount(); i++) {
  entry.setBigint(i, 1);
}
entry.setPartitionKey("TestPartitionKey");
recordEntries.add(entry);
// Write the data into DataHub.
PutRecordsResult result = client.putRecords("projectName", "topicName", recordEntries);
```

5.3.6. Create a DataConnector

5.3.6.1. Create a DataConnector for MaxCompute

The sample code is shown as follows:

```
public void createDataConnector () {
  // Create a DataConnector for MaxCompute
  // Configure the destination MaxCompute table.
  String odpsProject = "datahub_test";
  String odpsTable = "test_table";
  String odpsEndpoint = "http://service-all.ext.odps.example.com/api";
  String tunnelEndpoint = "http://dt-all.ext.odps.example.com";
  OdpsDesc odpsDesc = new OdpsDesc();
  odpsDesc.setProject(odpsProject);
  odpsDesc.setTable(odpsTable);
  odpsDesc.setOdpsEndpoint(odpsEndpoint);
  odpsDesc.setTunnelEndpoint(tunnelEndpoint);
  odpsDesc.setAccessId(accessId);
  odpsDesc.setAccessKey(accessKey);
  odpsDesc.setPartitionMode(OdpsDesc.PartitionMode.USER_DEFINE);
  // Select some or all columns in the topic to be archived to MaxCompute. The MaxCompute table must hav
e the same columns in the same order as selected.
  List<String> jobIds = new ArrayList<String>();
  columnFields.add("f1");
  // By default, the partition mode is set to UserDefine.
  // If you set the partition mode to SYSTEM_TIME or EVENT_TIME, you must complete the following configur
ation.
  // The configuration begins.
  int timeRange = 15;  // The partitioning interval, in minutes. Minimum value: 15.
  odpsDesc.setPartitionMode(OdpsDesc.PartitionMode.SYSTEM_TIME);
  odpsDesc.setTimeRange(timeRange);
  Map<String, String> partitionConfig = new LinkedHashMap<String, String>();
  // The partition key must be in the %Y%m%d%H%M format.
  partitionConfig.put("pt", "%Y%m%d");
  partitionConfig.put("ct", "%H%M");
  odpsDesc.setPartitionConfig(partitionConfig);
  // The configuration ends.
  client.createDataConnector(projectName, topicName, ConnectorType.SINK_ODPS, columnFields, odpsDe
sc);
  // You can obtain the DataConnector status periodically, such as every 15 minutes, to check whether all sh
ards are being archived properly to the destination platform.
```

```
  String shard = "0";
  GetDataConnectorShardStatusResult getDataConnectorShardStatusResult =
    client.getDataConnectorShardStatus(projectName, topicName, ConnectorType.SINK_ODPS, shard);
  System.out.println(getDataConnectorShardStatusResult.getCurSequence());
  System.out.println(getDataConnectorShardStatusResult.getLastErrorMessage());
}
```

5.3.6.2. Create a DataConnector for AnalyticDB or ApsaraDB RDS for MySQL

Sample code:

```
public void createADSDataConnector () {
  // Create a DataConnector for AnalyticDB or ApsaraDB RDS for MySQL.
  // Configure the destination table.
  String dbHost = "127.0.0.1";
  int dbPort = 3306;
  String dbName = "db";
  String user = "123";
  String password = "123";
  String tableName = "table";
  DatabaseDesc desc = new DatabaseDesc();
  desc.setHost(dbHost);
  desc.setPort(dbPort);
  desc.setDatabase(dbName);
  desc.setUser(user);
  desc.setPassword(password);
  desc.setTable(tableName);
  // The maximum size of records that can be written to DataHub at a time. Unit: Byte.
  desc.setMaxCommitSize(512L);
  // Specify whether to ignore errors and continue the process.
  desc.setIgnore(true);
  // Select several or all columns in the topic to be synchronized to AnalyticDB or ApsaraDB RDS for MySQL. T
he destination table must have the columns in the same order as selected.
  List<String> columnFields = new ArrayList<String>();
  columnFields.add("f1");
  client.createDataConnector(projectName, topicName, ConnectorType.SINK_ADS, columnFields, desc);
  // Alternative: client.createDataConnector(projectName, topicName, ConnectorType.SINK_MYSQL, colum
nFields, odpsDesc);
  // You can obtain the DataConnector status periodically, such as every 15 minutes, to check whether all sh
ards are being properly synchronized to the destination table.
  String shard = "0";
  GetDataConnectorShardStatusResult getDataConnectorShardStatusResult =
    client.getDataConnectorShardStatus(projectName, topicName, ConnectorType.SINK_ADS, shard);
  System.out.println(getDataConnectorShardStatusResult.getCurSequence());
  System.out.println(getDataConnectorShardStatusResult.getLastErrorMessage());
}
```

Note You cannot create DataConnectors for services that reside in virtual private clouds (VPCs) by using the SDK. You must create this type of DataConnectors in the DataHub console.

5.4. DataHub SDK for Python

5.4.1. Install DataHub SDK for Python

Run the following commands to install DataHub SDK for Python.

Quick installation

```
$ sudo pip install pydatahub
```

Install by using source code

```
$ git clone https://github.com/aliyun/aliyun-datahub-sdk-python.git
$ cd aliyun-datahub-sdk-python
$ sudo python setup.py install
```

Verify the installation

```
$ python -c "from datahub import DataHub"
```

5.4.2. Preparations

To access DataHub, you must have an account that is authorized to access DataHub, the AccessKey for the account, and an endpoint of DataHub. If you are using Apsara Stack V2, obtain the endpoint from the CMDB system. If you are using Apsara Stack V3, obtain the endpoint from the Apsara Infrastructure Management Framework console.

The following example shows how to initialize DataHub configurations:

```
import sys
import traceback
from datahub import DataHub
from datahub.utils import Configer
from datahub.models import Topic, RecordType, FieldType, RecordSchema, BlobRecord, TupleRecord, Curs
orType
from datahub.errors import DatahubException, ObjectAlreadyExistException
access_id = ***your access id***
access_key = ***your access key***
endpoint = ***your datahub server endpoint***
dh = DataHub(access_id, access_key, endpoint)
```

5.4.3. Create topics

Create a tuple topic

You must specify a schema for a tuple topic. The following data types are supported:

Tuple data types

Type	Description	Value range
Bigint	An 8-byte signed integer. **Note** Do not use the minimum value (-9223372036854775808) because this is a system reserved value.	-9223372036854775807 to 9223372036854775807
String	A string. Only UTF-8 encoding is supported.	The size of a string must not exceed 1 MB.
Boolean	One of two possible values.	Valid values: True and False, true and false, or 0 and 1.
Double	An 8-byte double-precision floating point.	-1.0 10^{308} to 1.0 10^{308}
TimeStamp	A timestamp.	It is accurate to milliseconds.

The sample code is shown as follows:

```
topic = Topic(name=topic_name)
topic.project_name = project_name
topic.shard_count = 3
topic.life_cycle = 7
topic.record_type = RecordType.TUPLE
topic.record_schema = RecordSchema.from_lists(['bigint_field', 'string_field', 'double_field', 'bool_field', 'ti
me_field'], [Fie
ldType.BIGINT, FieldType.STRING, FieldType.DOUBLE, FieldType.BOOLEAN, FieldType.TIMESTAMP])
try:
  dh.create_topic(topic)
  print "create topic success!"
  print "=======================================\n\n"
except ObjectAlreadyExistException, e:
  print "topic already exist!"
  print "=======================================\n\n"
except Exception, e:
  print traceback.format_exc()
  sys.exit(-1)
```

Create a blob topic

In a blob topic, a chunk of binary data is stored as a record. Records written into DataHub are Base64 encoded.

The sample code is shown as follows:

```
topic = Topic(name=topic_name)
topic.project_name = project_name
topic.shard_count = 3
topic.life_cycle = 7
topic.record_type = RecordType.BLOB
try:
  dh.create_topic(topic)
  print "create topic success!"
  print "=======================================\n\n"
except ObjectAlreadyExistException, e:
  print "topic already exist!"
  print "=======================================\n\n"
except Exception, e:
  print traceback.format_exc()
  sys.exit(-1)
```

5.4.4. Write records into DataHub and subscribe applications to records

Obtain the list of shards

You can use this SDK to obtain all shards in the topic.

```
shards = dh.list_shards(project_name, topic_name)
```

Note A list is returned and each entry in the list describes a shard in terms of the ID, status, starting hash key, and ending hash key.

Write records into DataHub

You can use this SDK to write records into a topic in DataHub.

```
failed_indexs = dh.put_records(project_name, topic_name, records)
```

Note The records request parameter is a list and each entry in the list represents a record. All records must be of the same data type, either tuple or blob. The subscript of the records that fail to be written into DataHub are returned.

The following sample code shows how to write tuple records into DataHub:

```
try:
    #Wait until all shards are in Active status.
    dh.wait_shards_ready(project_name, topic_name)
    print "shards all ready!!!"
    print "=========================================\n\n"
    topic = dh.get_topic(topic_name, project_name)
    print "get topic suc! topic=%s" % str(topic)
    if topic.record_type ! = RecordType.TUPLE:
        print "topic type illegal!"
        sys.exit(-1)
    print "=========================================\n\n"
    shards = dh.list_shards(project_name, topic_name)
    for shard in shards:
        print shard
    print "=========================================\n\n"
    records = []
    record0 = TupleRecord(schema=topic.record_schema, values=[1, 'yc1', 10.01, True, 1455869335000000])
    record0.shard_id = shards[0].shard_id
    record0.put_attribute('AK', '47')
    records.append(record0)
    record1 = TupleRecord(schema=topic.record_schema)
    record1['bigint_field'] = 2
    record1['string_field'] = 'yc2'
    record1['double_field'] = 10.02
    record1['bool_field'] = False
    record1['time_field'] = 1455869335000011
    record1.shard_id = shards[1].shard_id
    records.append(record1)
    record2 = TupleRecord(schema=topic.record_schema)
    record2['bigint_field'] = 3
    record2['string_field'] = 'yc3'
    record2['double_field'] = 10.03
    record2['bool_field'] = False
    record2['time_field'] = 1455869335000013
    record2.shard_id = shards[2].shard_id
    records.append(record2)
    failed_indexs = dh.put_records(project_name, topic_name, records)
    print "put tuple %d records, failed list: %s" %(len(records), failed_indexs)
    #If you specify the failed_indexs parameter, we recommend that you retry writing the failed records into D
```

```
ataHub.
  print "========================================\n\n"
except DatahubException, e:
  print traceback.format_exc()
  sys.exit(-1)
else:
  sys.exit(-1)
```

Obtain the cursor of a record

You can use this SDK to obtain the cursor of a record. Valid values: **OLDEST**, **LATEST**, and **SYSTEM_TIME**.

- OLDEST: the cursor that points to the earliest record in DataHub.
- LATEST: the cursor that points to the latest record in DataHub.
- SYSTEM_TIME: the cursor that points to the first record ingested after the specified time.

```
cursor = dh.get_cursor(project_name, topic_name, CursorType.OLDEST, shard_id)
```

> **Note** If you select SYSTEM_TIME, this SDK returns the cursor that points to the record ingested at the specified time. If no record is ingested at the specified time, the first record ingested after the time is returned.

Consume records

You can use this SDK to specify the cursor that points to a record where the consumption begins and specify the upper limit of records to be consumed. If fewer records exist than the specified limit, the actual number of records are returned.

```
dh.get_records(topic, shard_id, cursor, 10)
```

The following sample code shows how to consume tuple records:

```
try:
    #Wait until all shards are in Active status.
    dh.wait_shards_ready(project_name, topic_name)
    print "shards all ready!!!"
    print "========================================\n\n"
    topic = dh.get_topic(topic_name, project_name)
    print "get topic suc! topic=%s" % str(topic)
    if topic.record_type != RecordType.TUPLE:
        print "topic type illegal!"
        sys.exit(-1)
    print "========================================\n\n"
    cursor = dh.get_cursor(project_name, topic_name, CursorType.OLDEST, '0')
    while True:
        (record_list, record_num, next_cursor) = dh.get_records(topic, '0', cursor, 10)
        for record in record_list:
            print record
        if 0 == record_num:
            time.sleep(1)
        cursor = next_cursor
except DatahubException, e:
    print traceback.format_exc()
    sys.exit(-1)
else:
    sys.exit(-1)
```

5.5. Scalability

5.5.1. Scenarios

You can increase or decrease the number of shards in a topic according to the service load.

For example, if the topic throughput cannot handle a surge in the service load during Double 11, you can split existing shards to up to 256 to increase the throughput to 256 MB/s.

As the service load decreases after Double 11, you can reduce the number of shards as needed by performing the merge operation.

5.5.2. Obtain shard details

You can call the ListShard operation to obtain details about all shards.

The sample response is shown as follows:

```
{
  "ShardId": "string",
  "State": "string",
  "ClosedTime": uint64,
  "BeginHashKey": "string",
  "EndHashKey": "string",
  "ParentShardIds": [string,string,],
  "LeftShardId": "string",
  "RightShardId": "string"
}
```

5.5.3. Split a shard

You can split a shard by using an SDK or in the console. You can specify the shard by setting a shard ID and a 128-bit hash key value. After the shard is split into two child shards, the IDs and key values of the child shards are returned. The status of the parent shard is changed to Deactivated.

For example, before the split operation, the parent shard details are shown as follows:

```
ShardId:0 Status:ACTIVE BeginHashKey:00000000000000000000000000000000 EndHashKey:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Split the shard by using an SDK:

```
String shardId = "0";
SplitShardRequest req = new SplitShardRequest(projectName, topicName, shardId, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
SplitShardResult resp = client.splitShard(req);
```

After the split operation, the parent shard turns into the following three shards:

```
ShardId:0 Status:Deactivated BeginHashKey:00000000000000000000000000000000 EndHashKey:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
ShardId:1 Status:ACTIVE BeginHashKey:00000000000000000000000000000000 EndHashKey:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ShardId:2 Status:ACTIVE BeginHashKey:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA EndHashKey:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

5.5.4. Merge shards

You can merge two adjacent shards by using an SDK or in the console. Shards are considered adjacent if the union of the hash key ranges for the two shards forms a contiguous set with no gaps. After the two shards are merged, the ID and key value of the new shard are returned. The statuses of the two parent shards change to Deactivated.

For example, before the merge operation, the details of the two parent shards are as follows:

```
ShardId:0 Status:ACTIVE BeginHashKey:00000000000000000000000000000000 EndHashKey:7FFFFFFFFFFFFF7FFFFFFFFFFFFFFF
ShardId:1 Status:ACTIVE BeginHashKey:7FFFFFFFFFFFFFFF7FFFFFFFFFFFFFFF EndHashKey:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Merge the two shards by using an SDK:

```
String shardId = "0";
String adjacentShardId = "1";
MergeShardRequest req = new MergeShardRequest(projectName, topicName, shardId, adjacentShardId);
MergeShardResult resp = client.mergeShard(req);
```

After the merge operation, the parent shards turn into the following three shards:

```
ShardId:0 Status:Deactivated BeginHashKey:00000000000000000000000000000000 EndHashKey:7FFFFFFFFFFFFFF7FFFFFFFFFFFFFFF
ShardId:1 Status:Deactivated BeginHashKey:7FFFFFFFFFFFFFFF7FFFFFFFFFFFFFFF EndHashKey:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
ShardId:2 Status:ACTIVE BeginHashKey:00000000000000000000000000000000 EndHashKey:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

> **Note**
>
> - After merge or split operations are performed, the statuses of the parent shards change to Deactivated. New data cannot be written into deactivated shards, whereas existing data in deactivated shards can still be consumed. A deactivated shard cannot be split into two shards or merged with any other shards. When the time-to-live of the records in a deactivated shard expires, the shard is deleted.
> - If a DataConnector is configured for a deactivated shard, the DataConnector will be stopped after all records in the shard are copied to the destination platform. The DataConnector is automatically deleted after the shard is deleted.
> - The new shard can only be used after its status is changed to Active. It requires less than five seconds for the status to change to Active.

5.6. Insert a column to a topic schema

5.6.1. Insert a column

This section describes how to insert columns to a topic schema.

The sample code is shown as follows:

```
// Specify a DataHub account.
Account account = new AliyunAccount("your access id", "your access key");
DatahubConfiguration conf = new DatahubConfiguration(account, "datahub endpoint");
DatahubClient client = new DatahubClient(conf);
// Insert a bigint column to the topic schema.
client.appendField(new AppendFieldRequest(projectName, topicName, new Field("test", FieldType.BIGINT))
);
// If you have created a DataConnector for MaxCompute, call the following operation.
client.appendDataConnectorField(new AppendDataConnectorFieldRequest(projectName, topicName,  Con
nectorType.SINK_ODPS, "test"));
}
```

> **Note**
>
> - DataHub supports inserting columns only. Modifying or deleting columns is not supported.
> - The operation can only be performed on a DataConnector for MaxCompute: **Insert a column in MaxCompute > Insert a corresponding column in DataHub > Modify the corresponding DataConnector configuration.**

6.Graph Analytics

6.1. List of operations by function

Search

Operation	Description	URL
QueryNodes	Queries matched object nodes in a relation network. For example, you can query ID card information based on the registered household address, or query vehicle information based on the color and number of the license plate.	/rest/search/queryNodes.json
QueryLinks	Queries information about matched links in a relation network. For example, you can query the link between a person and a train number based on information such as the time of the ride and the departure station.	/rest/search/queryLinks.json

Relation network

Operation	Description	URL
GetFastGraph	Queries links related to an object node based on the default link.	/rest/graph/getFastGraph.json
GetGraph	Queries links related to one or more object nodes based on a specific link.	/rest/graph/getGraph.json
GetGroupGraph	Queries first-degree links related to multiple object nodes based on a specific link.	/rest/graph/getGroupGraph.json
CommonLinkGraph	Queries object nodes that have specific links with the specified object nodes.	/rest/graph/commonLinkGraph.json
PathAnalysisGraph	Obtains the shortest path between two object nodes or degree-specific links between two object nodes. You can query direct links with a maximum of six degrees.	/rest/graph/pathAnalysisGraph.json
KinshipGraph	Queries multi-degree links of the same type for an object node.	/rest/graph/kinshipGraph.json

Data persistence

Operation	Description	URL
PersistNode	Persists an object node that does not exist in a database of the system to the database.	/rest/etl/persistNode.json

Operation	Description	URL
PersistLink	Persists the link data between two object nodes. This operation persists your link data that does not exist in the system to a specific database.	/rest/etl/persistLink.json

OLEP escape

Operation	Description	URL
QueryOlpMetaConfig	Queries the configurations of the OLEP model in Graph Analytics. Users from other systems can call this operation to view service variable mappings.	/rest/meta/queryOlpMetaConfig.json
UpdateOlpMetaConfig	Updates the configuration of the OLEP model in Graph Analytics. Users from other systems can call this operation to configure service variable mappings.	/rest/meta/updateOlpMetaConfig.json

Collaboration and sharing

Operation	Description	URL
ShareAnalysis	Shares your analysis cases with specific users and analyzes these cases with them.	ShareAnalysis/rest/collaboration/shareAnalysis.json

Labels

Operation	Description	URL
AddLabels	Adds label information to objects or events so that you can locate and query them more easily.	/rest/label/addLabel.json
ObtainObjectLabels	Queries label information about the object or event that is being analyzed.	/rest/label/getObjShareLabels.json
LikeLabels	Adds likes to a label of an object or event.	/rest/label/incObjLabelCnt.json
CancelLikes	Cancels likes on a label of an object or event.	/rest/label/subObjLabelCnt.json
ModifyLabels	Modifies a label of an object or event.	/rest/label/editObjLabel.json
ObtainLabelHistory	Retrieves operation history related to a label, such as create, like, and unlike operations.	/rest/label/getObjLabelHistory.json
DeleteLabels	Deletes a label of an object or event.	/rest/label/delObjLabel.json

Pattern recognition

Operation	Description	URL
ObtainPatternsandDirectories	Obtains all patterns and directories, such as private patterns, public patterns, private directories, and public directories.	/rest/pattern/nav/list.json
CreateTasks	Creates pattern tasks. You can create multiple tasks for each pattern and can modify the parameters of each task.	/rest/pattern/task/create.json
QueryTaskStatus	Regularly queries the status of a task.	/rest/pattern/task/status.json
ViewTaskResults	Queries the results of a task. If the task ID is specified, the records of the specified task are returned. Otherwise, the latest records of the current user are returned.	/rest/pattern/task/result.json

6.2. Business restrictions

Business restrictions and resource constraints

If any part of the API instruction concerning optional parameter values or available specifications contradicts the specifications published on the official site, the information on the official site takes precedence.

Notes

The format of filter conditions that are critical to Graph Analytics:

```
{
      "propertyId":"L0001P001",
      "propertyType":"string_equal",
      "values":[""]
   }
```

Scenario 1

propertyType = *string_equal/string_like*, *string_equal/string_like*. You can specify multiple conditions. The logic between these conditions is OR.

Scenario 2

propertyType = *time/date/number*, *time/date/number* is an array of two values. For example,

- [10,20] indicates a value greater than or equal to 10 but less than or equal to 20.
- [10,10] indicates a value equal to 10.
- [10,*] indicates a value greater than or equal to 10.
- [*,20] indicates a value less than or equal to 20.

Scenario 3

propertyType = *dict_select*, *dict_select*. You can specify multiple conditions. The logic between these conditions is OR.

6.3. Make API requests

6.3.1. Request structures

Endpoints

Consult your PE based on the actual deployment environment.

Protocols

Graph Analytics uses HTTP as the communication protocol.

Methods

You can call an operation by sending an HTTP POST request to the Graph Analytics endpoint and adding relevant request parameters accordingly.

Request parameters

Graph Analytics provides request parameters in the JSON format using the application/json protocol, and the parameters are placed in requestBody.

Encoding

UTF-8 is used for character encoding.

6.3.2. Common parameters

Common request parameters

Graph Analytics supports service API request parameters in the JSON format and the parameters are placed in requestBody. The example is as follows:

```
{
   "objectType": "O0003",
   "objectvalue": [
    "O0003P0004-371411371416495795"
   ]
   }
```

Common response parameters

Graph Analytics supports service API response parameters in the JSON format. In the result returned by the query in the node data in the JSON structure, elapsedTime indicates the time when the operation is called for a query, noteMsg indicates the error information, and success indicates whether the operation is called successfully.

Examples

```
{
 "data": {
  "groups": [],
  "linkCnt": 0,
  "linkDetails": {},
  "linkPropMaps": {},
  "linkProps": {},
  "links": [],
  "nodeCnt": 0,
  "nodePropMaps": {},
  },
  "nodesProps": {}
 },
 "elapsedTime": 59,
  "errorcode": 0,
 "noteMsg": "",
 "success": true
}
```


6.3.3. Responses

Successful responses

Graph Analytics provides service API response parameters in the JSON format. A value of true indicates that the call is successful, and the successful result is stored in data.

For example:


```
{
 "data": {
 },
 "elapsedTime": 167,
 "errorcode": 0,
 "noteMsg": "",
 "success": true
}
```


Error responses

Graph Analytics provides service API response parameters in the JSON format. A value of false indicates that the call has failed, and the error message is stored in noteMsg.

For example:

```
{
 "data": {
 },
 "elapsedTime": 167,
 "noteMsg": "011005:The user has not logged on",
 "success": false
}
```

Common error codes

Error code	Error message	HTTP status code	Description
00XXXX	The error message is returned when a common system error occurs. The error message is described in noteMsg.	200	
10XXXX	The error message is returned when an authentication error occurs. The error message is described in noteMsg.	200	
20XXXX	The error message is returned when a user and role management error occurs. The error message is described in noteMsg.	200	
30XXXX	The error message is returned when a data source configuration error occurs. The error information is described in noteMsg.	200	
40XXXX	The error message is returned when a metadata configuration error occurs. The error message is described in noteMsg.	200	

Error code	Error message	HTTP status code	Description
50XXXX	The error message is returned when a cache refreshing error occurs. The error message is described in noteMsg.	200	
60XXXX	The error message is returned when a system parameter configuration error occurs. The error message is described in noteMsg.	200	
70XXXX	The error message is returned when a business parameter configuration error occurs. The error message is described in noteMsg.	200	

6.3.4. API restrictions

Request restrictions

To conserve computing resources and ensure the stability of Graph Analytics, each SID can receive a maximum of ten requests at the same time. If there are more than ten requests, an error message will be returned. Requests above the ten request threshold will be processed after a 10 second interval.

6.3.5. Authentication

In the case of initial calling or session time-out, the 011005 error code is returned, indicating that the user has no permission.

Authentication request

```
post http://ip:7001/login.json
{"userNum":"xianyi","password":"e10adc3949ba59abbe56e057f20f883e"}
```

Return:

```
{
 "data":{
  "cookie": "995282b4-82f3-444d-ac6a-50030912e243"····..
 },
 "elapsedTime": 167,
 "errorcode": 0,
 "noteMsg": "",
 "success": true
}
```


Verify authentication

For all follow-up requests, add cookie:sid=995282b4-82f3-444d-ac6a-50030912e243 in the head.

6.4. Search

6.4.1. QueryLinks

You can call this operation to query link information based on specific conditions. For example, you can query the link between a person and a train number based on the departure time and the departure station.

Request parameters

Parameter	Type	Required	Description
linkType	String	Yes	The type of the link node. It matches the link type configured in Administration Console. For example, O0001 indicates a call number.
propertyList	Array< PropertyFilter>	Yes	
propertyId	String	No	The link property used for filtering.
propertyType	String	No	The type of the link property used for filtering.
values	Array<String>	No	The value of the link property used for filtering.

Response parameters

Parameter	Type	Description
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.
linkDetails	Array<String>	The ID list of link details contained in a link.
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.
<Property>	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests

```
{
  "linkType":"L0001",
  "propertyList":[
    {
      "propertyId":"L0001P001",
      "propertyType":"string_equal",
      "values":[""]
    }
  ]
}
```

Sample responses


```
{
 "data": {
  "linkCnt": 2,
  "linkDetails": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "label": "By train",
    "linkId": "L0003^27b44b00fcf663b8fdfad968f8a2eba9",
    "linkType": "L0003",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "label": "By train",
    "linkId": "L0003^49052e268251cf1b8252c407961e132a",
    "linkType": "L0003"
   }
  },
  "linkProps": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "L0003P0001":"",
    "L0003P0002":"",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "L0003P0001":"",
    "L0003P0002":"",
   }
  },
  "links": [
   {
    "id": "O0003#O0003P0004-893751893750806906^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^27b44b00fcf663b8fdfad968f8a2eba9"
    ],
    "source": "O0003P0004-893751893750806906",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   },
   {
    "id": "O0003#O0003P0004-3714113714164957 95^O0004#O0004P0002-HB1163",
    "linkDetails": [
```

```
      "L0003^49052e268251cf1b8252c407961e132a"
    ],
    "source": "O0003P0004-3714113714164957 95",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   }
  ],
 },
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```


6.4.2. QueryNodes

You can call this operation to query the matched object nodes in the relationship network. For example, you can query the ID card information using the registered household address, or query vehicle information by the color of the license plate and the license plate number.

Request parameters

Name	Type	Required	Description
objectType	String	Yes	The type of the object node. It matches the object type configured in Administration Console. For example, O0001 indicates a mobile phone number.
propertyList	Array< PropertyFilter>	Yes	The properties used to filter objects.
propertyId	String	No	The object property used for filtering.
propertyType	String	No	The type of the object property used for filtering.
values	Array<String>	No	The values of the object properties used for filtering.

Response parameters

Name	Type	Description
Nodes	Array<Node>	The node list.
id	String	The ID of the node.

Name	Type	Description
label	String	The label of the node. It is consistent with the object property configured in Administration Console.
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	<Property>	The node property list.
<Property>	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".

Sample requests

```
{
  "objectType":"O0001",
  "propertyList":[
    {
      "propertyId":"O0001P001",
      "propertyType":"string_equal"
      "values":["321321321"]
    }
  ]
   }
```

Sample responses

```
{
 "data": {
  "nodes": {
   "O0003P0004-893751893750806906": {
    "id": "O0003P0004-893751893750806906",
    "label": "John",
    "type": "O0003",
    "virtual": false
   }
  },
  "nodesProps": {
   "O0003P0004-893751893750806906": {
    "O0003P0001":"John",
    "O0003P0002":"Male",
    "O0003P0003":"Zhejiang Province",
    "O0003P0004":"893751893750806906"
   }
  }
 },
 "elapsedTime": 50,
 "noteMsg": "",
 "success": true
    }
```


6.5. Relationship network

6.5.1. GetFastGraph

You can call this operation to start from one node to query specific links that are related to the node by default.

- The default link is configured in Administration Console. You can configure the link type, and the link properties and target node properties to be filtered.
- If the default link is not configured, all direct links will be queried by default.

Request parameters

Parameter	Type	Required	Description
objectType	String	Yes	The type of the object node. It matches the object type configured in Administration Console. For example, O0001 indicates a mobile phone number.

Parameter	Type	Required	Description
objectValue	String	Yes	The property type and value of the primary key of the object. The property type is consistent with the object primary key type configured in Administration Console. The primary key and key value are connected with a hyphen (-). For example, ID card-ID No., O0003P0004-371411371416495795.
degree	int	Yes	The supported query degree. You can query up to the fifth degree. We do not recommend any query beyond the fifth degree.

Response parameters

Parameter	Type	Description
nodeCnt	Integer	The number of object nodes on a network.
linkCnt	Integer	The number of links on a network.
nodes	Array<Node>	The node list.
id	String	The ID of the node.
label	String	The label of the node. It is consistent with the object property configured in Administration Console.
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	Array<Property>	The node property list.
<Property>	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.

Parameter	Type	Description
linkDetails	Array<String>	The ID list of link details contained in a link.
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.
<Property>	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests

```
{
 "workspaceId":"",
 "objectType":"O0001",
 "objectValue":"O0001P001",
 "degree":3,
 "defaultTimeCond":{
  "propertyId":"",
  "propertyType":"time",
  "values":[null,null]
  }
}
```

Sample responses

```
{
 "data": {
  "linkCnt": 2,
  "linkDetails": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "label": "By train",
    "linkId": "L0003^27b44b00fcf663b8fdfad968f8a2eba9",
    "linkType": "L0003",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "label": "By train",
```

```
    "linkId": "L0003^49052e268251cf1b8252c407961e132a",
    "linkType": "L0003"
   }
  },
  "linkProps": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "L0003P0001":"",
    "L0003P0002":"",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "L0003P0001":"",
    "L0003P0002":"",
   }
  },
  "links": [
   {
    "id": "O0003#O0003P0004-893751893750806906^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^27b44b00fcf663b8fdfad968f8a2eba9"
    ],
    "source": "O0003P0004-893751893750806906",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   },
   {
    "id": "O0003#O0003P0004-3714113714164957 95^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^49052e268251cf1b8252c407961e132a"
    ],
    "source": "O0003P0004-3714113714164957 95",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   }
  ],
  "nodeCnt": 3,
  "nodes": {
   "O0003P0004-3714113714164957 95": {
    "id": "O0003P0004-3714113714164957 95",
```

```
      "label": "Jane",
      "type": "O0003",
      "virtual": false
    },
    "O0003P0004-893751893750806906": {
      "id": "O0003P0004-893751893750806906",
      "label": "John",
      "type": "O0003",
      "virtual": false
    },
    "O0004P0002-HB1163": {
      "id": "O0004P0002-HB1163",
      "label": "HB1163",
      "type": "O0004",
      "virtual": false
    }
  },
  "nodesProps": {
    "O0003P0004-371411371416495795": {
      "O0003P0001":"Jane",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"371411371416495795"
    },
    "O0003P0004-893751893750806906": {
      "O0003P0001":"John",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"893751893750806906"
    },
    "O0004P0002-HB1163": {
      "O0003P0001":"",
      "O0003P0002":"HB1163",
      "O0003P0003":""
    }
  },
  "elapsedTime": 0,
  "noteMsg": "",
  "success": true
}
```

6.5.2. GetGraph

You can call this operation to start from multiple nodes to query specific links and nodes that are directly related to the input node.

- You can query from nodes of different types.
- You can query multiple links.
- You can define filter conditions for link properties.
- You can define filter conditions for target node properties.

Request parameters

Parameter	Type	Required	Description
objects	String,Array<String>	Yes	The list of start nodes in the KV structure. K indicates the object type, and it is consistent with the configuration in Administration Console. V indicates the object ID array. For more information, see the sample request.
links	Array<Link>	Yes	The links to be queried in the graph query.
linkType	String	Yes	The link type. It is consistent with the configuration in Administration Console.
direct	int	No	The link directions. 0: outbound, 1: inbound, 2: bidirectional, 3: undirected. If the parameter is not specified, no direction is used by default. Link lookup, group analysis, and common neighbor analysis are supported.
queryProps	Array<String>	No	The properties to be queried. By default, all properties are queried.
propertyList	Array<PropertyFilter>	No	The properties used to filter links. For more information, see the example.
propertyId	String	No	The link property used for filtering.
propertyType	String	No	The query type of the link property.
values	Array<String>	No	For more information, see **Notes** in Business restrictions.

Parameter	Type	Required	Description
derivated	Array<PropertyFilter>	No	The properties of the same word class that are used for filtering. For more information, see the example. The word class is the indirect link configured and managed in Administration Console. If the parameter is not specified, the default configuration is used.
statPropList	Array<statPropFilter>	No	The statistical property conditions used to filter sum and count values of numeric data types. The property is configured in Administration Console. Link lookup, group analysis, and common neighbor analysis are supported.
propertyId	string	No	A fixed value. In most cases, the format is: linkType+T+basic property ID (P000 for count). For example, L0001TP000.
propertyType	string	No	Fixed to number.
values	List<String>	No	This parameter is used in the same way as basic properties.
statType	int	No	The statistical condition type. The value 0 indicates querying by the property value or the number of records. The value 1 indicates querying by the property value or the order of records.
targetObjects	Array<ObjectFilter>	No	Properties used to filter the target objects. For more information, see the example.
objectType	String	Yes	The node type. It is consistent with the configuration in Administration Console.
propertyList	Array<PropertyFilter>	No	The property that requires the target node. For more information, see the example.
propertyId	String	No	The target node property used for filtering.
propertyType	String	No	The type of the target node property used for filtering.

Parameter	Type	Required	Description
values	Array<String>	No	The values of the target node properties used for filtering.

Response parameters

Parameter	Type	Description
nodeCnt	Integer	The number of object nodes on a network.
linkCnt	Integer	The number of links on a network.
nodes	Array<Node>	The node list.
id	String	The ID of the node.
label	String	The label of the node. It is consistent with the object property configured in Administration Console.
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	Array<Property>	The node property list.
<Property>	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.
linkDetails	Array<String>	The ID list of link details contained in a link.
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.

Parameter	Type	Description
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.
<Property>	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests

```
{
 "data":{
  "objects":{
   "O0003":[
    "O0003P0004-371411371416495795",
    "O0003P0004-893751893750806906"
   ],
   "O0004":[
     "O0004P0002-HB1163",
     "O0004P0002-HB1985",
     "O0004P0002-HB4999"
   ]
  },
  "links":[
   {
    "linkType":"L0003",
    "propertyList":[],
    "derivated":[]
   },
   {
    "linkType":"L0004",
    "propertyList":[
     {
      "propertyId":"L0004CC002",
      "propertyType":"string_like",
      "values":[
       "11"
      ]
```

```
      }
     ],
     "derivated": [
      {
       "count": {
        "propertyType": "string_equal",
        "values": [
         "1"
        ]
       },
       "propIds": [
        "L0004CC002"
       ],
       "intervaltime": []
      }
     ]
    }
   ],
   "targetObjects": [
    {
     "objectType": "O0003",
     "propertyList": [
      {
       "propertyId": "O0003P0002",
       "propertyType": "string_like",
       "values": [
        "22"
       ]
      }
     ]
    }
   ]
  }
}
```

Sample responses

```
{
 "data": {
  "linkCnt": 2,
  "linkDetails": {
```

```
  "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
   "label": "By train",
   "linkId": "L0003^27b44b00fcf663b8fdfad968f8a2eba9",
   "linkType": "L0003",
  },
  "L0003^49052e268251cf1b8252c407961e132a": {
    "label": "By train",
   "linkId": "L0003^49052e268251cf1b8252c407961e132a",
   "linkType": "L0003"
  }
 },
 "linkProps": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
     "L0003P0001":"",
     "L0003P0002":"",
   } ,
   "L0003^49052e268251cf1b8252c407961e132a": {
     "L0003P0001":"",
     "L0003P0002":"",
   }
 },
 "links": [
  {
   "id": "O0003#O0003P0004-893751893750806906^O0004#O0004P0002-HB1163",
   "linkDetails": [
    "L0003^27b44b00fcf663b8fdfad968f8a2eba9"
   ],
   "source": "O0003P0004-893751893750806906",
   "sourceType": "O0003",
   "target": "O0004P0002-HB1163",
   "targetType": "O0004"
  },
  {
   "id": "O0003#O0003P0004-3714113714164957 95^O0004#O0004P0002-HB1163",
   "linkDetails": [
    "L0003^49052e268251cf1b8252c407961e132a"
   ],
   "source": "O0003P0004-3714113714164957 95",
   "sourceType": "O0003",
   "target": "O0004P0002-HB1163",
   "targetType": "O0004"
```

```
    "targetType": "O0004"
   }
  ],
  "nodeCnt": 3,
  "nodes": {
   "O0003P0004-371411371416495795": {
    "id": "O0003P0004-371411371416495795",
    "label": "Jane",
    "type": "O0003",
    "virtual": false
   },
   "O0003P0004-893751893750806906": {
    "id": "O0003P0004-893751893750806906",
    "label": "John",
    "type": "O0003",
    "virtual": false
   },
   "O0004P0002-HB1163": {
    "id": "O0004P0002-HB1163",
    "label": "HB1163",
    "type": "O0004",
    "virtual": false
   }
  },
  "nodesProps": {
    "O0003P0004-371411371416495795": {
      "O0003P0001":"Jane",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"371411371416495795"
    },
    "O0003P0004-893751893750806906": {
      "O0003P0001":"John",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"893751893750806906"
    },
    "O0004P0002-HB1163": {
      "O0003P0001":"",
      "O0003P0002":"HB1163",
      "O0003P0003":""
```

```
    }
  }
 },
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```

6.5.3. GetGroupGraph

You can call this operation to start from multiple nodes to query the first-degree paths of a specific link.

- You can query from nodes of different types.
- You can query multiple links.
- You can define the filter conditions for link properties.
- The target nodes: Group analysis only queries the nodes you have input.

Request parameters

Parameter	Type	Required	Description
group	Array<group>	No	The node group used in the group analysis for merged nodes.
id	String	No	The ID of the merged node. Its value must be unique.
children	Array<String>	No	The nodes covered under the ID of the merged node.
objects	String,Array<String>	Yes	The list of start nodes in the KV structure. K indicates the object type, and it is consistent with the configuration in Administration Console. V indicates the object ID array. For more information, see the sample request.
links	Array<Link>	Yes	The links to be queried in the graph query.
linkType	String	Yes	The link type. It is consistent with the configuration in Administration Console.

Parameter	Type	Required	Description
direct	int	No	The link directions. 0: outbound, 1: inbound, 2: bidirectional, 3: undirected. If the parameter is not specified, no direction is used by default. Link lookup, group analysis, and common neighbor analysis are supported.
queryProps	Array<String>	No	The properties to be queried. By default, all properties are queried.
propertyList	Array<PropertyFilter>	No	The properties used to filter links. For more information, see the example.
propertyId	String	No	The link property used for filtering.
propertyType	String	No	The query type of the link property.
values	Array<String>	No	For more information, see **Notes** in Business restrictions.
derivated	Array<PropertyFilter>	No	The properties of the same word class that are used for filtering. For more information, see the example. The word class is the indirect link configured and managed in Administration Console. If the parameter is not specified, the default configuration is used.
statPropList	Array<statPropFilter>	No	The statistical property conditions used to filter sum and count values of numeric data types. The property is configured in Administration Console. Link lookup, group analysis, and common neighbor analysis are supported.
propertyId	string	No	A fixed value. In most cases, the format is: linkType+T+basic property ID (P000 for count). For example, L0001TP000.
propertyType	string	No	Fixed to number.
values	List<String>	No	This parameter is used in the same way as basic properties.

Parameter	Type	Required	Description
statType	int	No	The statistical condition type. The value 0 indicates querying by the property value or the number of records, and the value 1 indicates querying by the property value or the order of records.

Response parameters

Parameter	Type	Description
nodeCnt	Integer	The number of object nodes on a network.
linkCnt	Integer	The number of links on a network.
nodes	Array<Node>	The node list.
id	String	The ID of the node.
label	String	The label of the node. It is consistent with the object property configured in Administration Console.
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	Array<Property>	The node property list.
<Property>	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.

Parameter	Type	Description
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.
linkDetails	Array<String>	The ID list of link details contained in a link.
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.
<Property>	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests

```
{
 "data":{
  "objects":{
   "O0003":[
    "O0003P0004-371411371416495795",
    "O0003P0004-893751893750806906"
   ],
   "O0004":[
    "O0004P0002-HB1163",
    "O0004P0002-HB1985",
    "O0004P0002-HB4999"
   ]
  },
  "links":[
   {
    "linkType":"L0003",
    "propertyList":[],
```

```
    "derivated": []
   },
   {
    "linkType": "L0004",
    "propertyList": [
     {
      "propertyId": "L0004CC002",
      "propertyType": "string_like",
      "values": [
       "11"
      ]
     }
    ],
    "derivated": [
     {
      "count": {
       "propertyType": "string_equal",
       "values": [
        "1"
       ]
      },
      "propIds": [
       "L0004CC002"
      ],
      "intervaltime": []
     }
    ]
   }
  ],
  "targetObjects": [
   {
    "objectType": "O0003",
    "propertyList": [
     {
      "propertyId": "O0003P0002",
      "propertyType": "string_like",
      "values": [
       "22"
      ]
     }
    ]
```

```
    }
   ]
  }
}
```

Sample responses


```
{
 "data": {
  "linkCnt": 2,
  "linkDetails": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "label": "By train",
    "linkId": "L0003^27b44b00fcf663b8fdfad968f8a2eba9",
    "linkType": "L0003",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "label": "By train",
    "linkId": "L0003^49052e268251cf1b8252c407961e132a",
    "linkType": "L0003"
   }
  },
  "linkProps": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "L0003P0001":"",
    "L0003P0002":"",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "L0003P0001":"",
    "L0003P0002":"",
   }
  },
  "links": [
   {
    "id": "O0003#O0003P0004-893751893750806906^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^27b44b00fcf663b8fdfad968f8a2eba9"
    ],
    "source": "O0003P0004-893751893750806906",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
```

```
    "targetType": "O0004"
   },
   {
    "id": "O0003#O0003P0004-371411371416495795^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^49052e268251cf1b8252c407961e132a"
    ],
    "source": "O0003P0004-371411371416495795",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   }
  ],
  "nodeCnt": 3,
  "nodes": {
   "O0003P0004-371411371416495795": {
    "id": "O0003P0004-371411371416495795",
    "label": "Jane",
    "type": "O0003",
    "virtual": false
   },
   "O0003P0004-893751893750806906": {
    "id": "O0003P0004-893751893750806906",
    "label": "John",
    "type": "O0003",
    "virtual": false
   },
   "O0004P0002-HB1163": {
    "id": "O0004P0002-HB1163",
    "label": "HB1163",
    "type": "O0004",
    "virtual": false
   }
  },
  "nodesProps": {
   "O0003P0004-371411371416495795": {
    "O0003P0001":"Jane",
    "O0003P0002":"",
    "O0003P0003":"",
    "O0003P0004":"371411371416495795"
```

```
    },
    "O0003P0004-893751893750806906": {
     "O0003P0001":"John",
     "O0003P0002":"",
     "O0003P0003":"",
     "O0003P0004":"893751893750806906"
    },
    "O0004P0002-HB1163": {
     "O0003P0001":"",
     "O0003P0002":"HB1163",
     "O0003P0003":""
    }
   }
  },
  "elapsedTime": 0,
  "noteMsg": "",
  "success": true
}
```


6.5.4. CommonLinkGraph

You can call this operation to query object nodes that have a specific relationship with these object nodes, starting from multiple object nodes.

- You can query from nodes of different types.
- You can query multiple links.
- You can define the filter conditions for link properties.
- You can define the filter conditions for target nodes.
- You can configure the number of common neighbors.

Request parameters

Parameter	Type	Required	Description
objects	String,Array<String>	Yes	The list of start nodes in the KV structure. K indicates the object type, and it is consistent with the configuration in Administration Console. V indicates the object ID array. For more information, see the sample request.
links	Array<Link>	Yes	The links to be queried in the graph query.

Parameter	Type	Required	Description
linkType	String	Yes	The link type. It is consistent with the configuration in Administration Console.
direct	int	No	The link directions. 0: outbound, 1: inbound, 2: bidirectional, 3: undirected. If the parameter is not specified, no direction is used by default. Link lookup, group analysis, and common neighbor analysis are supported.
queryProps	Array<String>	No	The properties to be queried. By default, all properties are queried.
propertyList	Array<PropertyFilter>	No	The properties used to filter links. For more information, see the example.
propertyId	String	No	The link property used for filtering.
propertyType	String	No	The query type of the link property.
values	Array<String>	No	For more information, see **Notes** in Business restrictions.
derivated	Array<PropertyFilter>	No	The properties of the same word class that are used for filtering. For more information, see the example. The word class is the indirect link configured and managed in Administration Console. If the parameter is not specified, the default configuration is used.
statPropList	Array<statPropFilter>	No	The statistical property conditions used to filter sum and count values of numeric data types. The property is configured in Administration Console. Link lookup, group analysis, and common neighbor analysis are supported.
propertyId	string	No	A fixed value. In most cases, the format is: linkType+T+basic property ID (P000 for count). For example, L0001TP000.
propertyType	string	No	Fixed to number.
values	List<String>	No	This parameter is used in the same way as basic properties.

Parameter	Type	Required	Description
statType	int	No	The statistical condition type. The value 0 indicates querying by the property value or the number of records. The value 1 indicates querying by the property value or the order of records.
targetObjects	Array<ObjectFilter>	No	Properties used to filter the target objects. For more information, see the example.
objectType	String	Yes	The node type. It is consistent with the configuration in Administration Console.
propertyList	Array<PropertyFilter>	No	The property that requires the target node. For more information, see the example.
propertyId	String	No	The target node property used for filtering.
propertyType	String	No	The type of the target node property used for filtering.
values	Array<String>	No	The values of the target node properties used for filtering.
degree	int	No	Used for path analysis, common neighbor service, and lineage analysis.

Response parameters

Parameter	Type	Description
nodeCnt	Integer	The number of object nodes on a network.
linkCnt	Integer	The number of links on a network.
nodes	Array<Node>	The node list.
id	String	The ID of the node.
label	String	The label of the node. It is consistent with the object property configured in Administration Console.

Parameter	Type	Description
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	Array<Property>	The node property list.
<Property>	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.
linkDetails	Array<String>	The ID list of link details contained in a link.
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.
<Property>	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests


```
{
 "data": {
  "degree": "3",
  "objects": {
   "O0003": [
    "O0003P0004-371411371416495795",
    "O0003P0004-893751893750806906"
   ],
   "O0004":[
    "O0004P0002-HB1163",
    "O0004P0002-HB1985",
    "O0004P0002-HB4999"
   ]
  },
  "links": [
   {
    "linkType": "L0003",
    "propertyList": [],
    "derivated": []
   },
   {
    "linkType": "L0004",
    "propertyList": [
     {
      "propertyId": "L0004CC002",
      "propertyType": "string_like",
      "values": [
       "11"
      ]
     }
    ],
    "derivated": [
     {
      "count": {
       "propertyType": "string_equal",
       "values": [
        "1"
       ]
      },
```

```
        "propIds": [
          "L0004CC002"
        ],
        "intervaltime": []
      }
    ]
   }
  ],
  "targetObjects": [
   {
    "objectType": "O0003",
    "propertyList": [
     {
      "propertyId": "O0003P0002",
      "propertyType": "string_like",
      "values": [
       "22"
      ]
     }
    ]
   }
  ]
 }
}
```

Sample responses

```
{
 "data": {
  "linkCnt": 2,
  "linkDetails": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "label": "By train",
    "linkId": "L0003^27b44b00fcf663b8fdfad968f8a2eba9",
    "linkType": "L0003",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "label": "By train",
    "linkId": "L0003^49052e268251cf1b8252c407961e132a",
    "linkType": "L0003"
   }
```

```
},
"linkProps": {
 "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
  "L0003P0001":"",
  "L0003P0002":"",
 },
 "L0003^49052e268251cf1b8252c407961e132a": {
  "L0003P0001":"",
  "L0003P0002":"",
 }
},
"links": [
 {
  "id": "O0003#O0003P0004-893751893750806906^O0004#O0004P0002-HB1163",
  "linkDetails": [
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9"
  ],
  "source": "O0003P0004-893751893750806906",
  "sourceType": "O0003",
  "target": "O0004P0002-HB1163",
  "targetType": "O0004"
 },
 {
  "id": "O0003#O0003P0004-371411371416495795^O0004#O0004P0002-HB1163",
  "linkDetails": [
   "L0003^49052e268251cf1b8252c407961e132a"
  ],
  "source": "O0003P0004-371411371416495795",
  "sourceType": "O0003",
  "target": "O0004P0002-HB1163",
  "targetType": "O0004"
 }
],
"nodeCnt": 3,
"nodes": {
 "O0003P0004-371411371416495795": {
  "id": "O0003P0004-371411371416495795",
  "label": "Jane",
  "type": "O0003",
  "virtual": false
```

```
    },
    "O0003P0004-893751893750806906": {
      "id": "O0003P0004-893751893750806906",
      "label": "John",
      "type": "O0003",
      "virtual": false
    },
    "O0004P0002-HB1163": {
      "id": "O0004P0002-HB1163",
      "label": "HB1163",
      "type": "O0004",
      "virtual": false
    }
  },
  "nodesProps": {
    "O0003P0004-371411371416495795": {
      "O0003P0001":"Jane",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"371411371416495795"
    },
    "O0003P0004-893751893750806906": {
      "O0003P0001":"John",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"893751893750806906"
    },
    "O0004P0002-HB1163": {
      "O0003P0001":"",
      "O0003P0002":"HB1163",
      "O0003P0003":""
    }
  }
 },
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```


6.5.5. PathAnalysisGraph

You can call this operation to obtain the shortest network path between two nodes or paths of a specific degree. You can query direct links with a maximum of six degrees.

- You can query from nodes of different types.
- You can query multiple links.
- You can define the filter conditions for link properties.
- You can define filter conditions for target nodes.
- You can configure the number of paths to be queried for the path analysis.
- You can query a maximum of six degrees of relationship extension.
- Path analysis does not support indirect relationships.

Request parameters

Name	Type	Required	Description
objects	String, Array<String>	Yes	The list of start nodes in the KV structure. K indicates the object type, and it is consistent with the configuration in Administration Console. V indicates the object ID array.
links	Array<Link>	Yes	The specified link for the graph query.
linkType	String	Yes	The link type. It is consistent with the configuration in Administration Console.
direct	int	No	The link direction. • 0: outbound • 1: inbound • 2: bidirectional • 3: undirected • If the parameter is not specified, no direction is used by default. Link lookup, group analysis, and common neighbor analysis are supported.
queryProps	Array<String>	No	The properties to be queried. By default, all properties are queried.
propertyList	Array<PropertyFilter>	No	The properties used to filter links. For more information, see the example.
propertyId	String	No	The link property used for filtering.
propertyType	String	No	The query type of the link property.

Name	Type	Required	Description
values	Array<String>	No	For more information, see **Notes** in Business restrictions.
degree	int	No	If the value is specified, all paths within the specified degree are returned. If the value is not specified, the shortest path is queried by default.

Response parameters

Name	Type	Description
nodeCnt	Integer	The number of object nodes on a network.
linkCnt	Integer	The number of links on a network.
nodes	Array<Node>	The node list.
id	String	The ID of the node.
label	String	The label of the node. It is consistent with the object property configured in Administration Console.
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	Array<Property>	The node property list.
<Property>	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.
linkDetails	Array<String>	The ID list of link details contained in a link.

Name	Type	Description
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.
<Property>	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests

```
{
 "data":{
  "type":"remote",
  "degree":"3",
  "objects":{
   "O0003":[
    "O0003P0004-371411371416495795",
    "O0003P0004-893751893750806906"
   ],
   "O0004":[
    "O0004P0002-HB1163",
    "O0004P0002-HB1985",
    "O0004P0002-HB4999"
   ]
  },
  "links":[
   {
    "linkType":"L0003",
    "propertyList":[],
    "derivated":[]
   },
   {
    "linkType":"L0004",
    "propertyList":[
     {
```

```
        "propertyId": "L0004CC002",
        "propertyType": "string_like",
        "values": [
         "11"
        ]
       }
      ],
      "derivated": [
       {
        "count": {
         "propertyType": "string_equal",
         "values": [
          "1"
         ]
        },
        "propIds": [
         "L0004CC002"
        ],
        "intervaltime": []
       }
      ]
     }
    ],
    "targetObjects": [
     {
      "objectType": "O0003",
      "propertyList": [
       {
        "propertyId": "O0003P0002",
        "propertyType": "string_like",
        "values": [
         "22"
        ]
       }
      ]
     }
    ]
   }
}
```

Sample responses


```
{
 "data": {
  "linkCnt": 2,
  "linkDetails": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "label": "By train",
    "linkId": "L0003^27b44b00fcf663b8fdfad968f8a2eba9",
    "linkType": "L0003",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "label": "By train",
    "linkId": "L0003^49052e268251cf1b8252c407961e132a",
    "linkType": "L0003"
   }
  },
  "linkProps": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "L0003P0001":"",
    "L0003P0002":"",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
    "L0003P0001":"",
    "L0003P0002":"",
   }
  },
  "links": [
   {
    "id": "O0003#O0003P0004-893751893750806906^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^27b44b00fcf663b8fdfad968f8a2eba9"
    ],
    "source": "O0003P0004-893751893750806906",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   },
   {
    "id": "O0003#O0003P0004-3714113714164957 95^O0004#O0004P0002-HB1163",
    "linkDetails": [
```

```
      "L0003^49052e268251cf1b8252c407961e132a"
    ],
    "source": "O0003P0004-371411371416495795",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
  }
 ],
 "nodeCnt": 3,
 "nodes": {
  "O0003P0004-371411371416495795": {
    "id": "O0003P0004-371411371416495795",
    "label": "Jane",
    "type": "O0003",
    "virtual": false
  },
  "O0003P0004-893751893750806906": {
    "id": "O0003P0004-893751893750806906",
    "label": "John",
    "type": "O0003",
    "virtual": false
  },
  "O0004P0002-HB1163": {
    "id": "O0004P0002-HB1163",
    "label": "HB1163",
    "type": "O0004",
    "virtual": false
  }
 },
 "nodesProps": {
  "O0003P0004-371411371416495795": {
   "O0003P0001":"Jane",
   "O0003P0002":"",
   "O0003P0003":"",
   "O0003P0004":"371411371416495795"
  },
  "O0003P0004-893751893750806906": {
   "O0003P0001":"John",
   "O0003P0002":"",
   "O0003P0003":"",
   "O0003P0004":"893751893750806906"
```

```
    },
    "O0004P0002-HB1163": {
     "O0003P0001":"",
     "O0003P0002":"HB1163",
     "O0003P0003":""
    }
   }
  },
  "elapsedTime": 0,
  "noteMsg": "",
  "success": true
}
```


6.5.6. KinshipGraph

This operation involves multi-degree links. You can call this operation to perform multi-degree queries on same-type links that are related to one node.

- A single object is supported.
- To pre-configure a specific link, log on to Administration Console. Choose **Advanced > System Settings > Lineage Analysis Settings**. Typically, the links to be pre-configured are lineage relationship and registered household number.
- Conditional filtering is not supported.
- You can configure the query degree. The maximum query degree is five.

Request parameters

Name	Type	Required	Description
objects	String,Array<String>	Yes	The list of start nodes in the KV structure. K indicates the object type, and it is consistent with the configuration in Administration Console. V indicates the object ID array.
degree	int	No	If the value is specified, all paths within the specified degree are returned. If the value is not specified, the shortest path is queried by default.

Response parameters

Name	Type	Description

Name	Type	Description
nodeCnt	Integer	The number of object nodes on a network.
linkCnt	Integer	The number of links on a network.
nodes	Array<Node>	The node list.
id	String	The ID of the node.
label	String	The node label. It is consistent with the object property configured in Administration Console.
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	Array<Property>	The node property list.
Property	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.
linkDetails	Array<String>	The ID list of link details contained in a link.
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.

Name	Type	Description
Property	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests

```
{
  "degree": "3",
  "objects": {
   "O0003": [
    "O0003P0004-371411371416495795"
   ]
  }
}
```

Sample responses

```
{
 "data": {
  "linkCnt": 2,
  "linkDetails": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
    "label": "By train",
    "linkId": "L0003^27b44b00fcf663b8fdfad968f8a2eba9",
    "linkType": "L0003",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
     "label": "By train",
    "linkId": "L0003^49052e268251cf1b8252c407961e132a",
    "linkType": "L0003"
   }
  },
  "linkProps": {
   "L0003^27b44b00fcf663b8fdfad968f8a2eba9": {
     "L0003P0001":"",
     "L0003P0002":"",
   },
   "L0003^49052e268251cf1b8252c407961e132a": {
     "L0003P0001":"",
```

```
      "L0003P0002":"",
    }
  },
  "links": [
   {
    "id": "O0003#O0003P0004-893751893750806906^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^27b44b00fcf663b8fdfad968f8a2eba9"
    ],
    "source": "O0003P0004-893751893750806906",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   },
   {
    "id": "O0003#O0003P0004-371411371416495795^O0004#O0004P0002-HB1163",
    "linkDetails": [
     "L0003^49052e268251cf1b8252c407961e132a"
    ],
    "source": "O0003P0004-371411371416495795",
    "sourceType": "O0003",
    "target": "O0004P0002-HB1163",
    "targetType": "O0004"
   }
  ],
  "nodeCnt": 3,
  "nodes": {
   "O0003P0004-371411371416495795": {
    "id": "O0003P0004-371411371416495795",
    "label": "Jane",
    "type": "O0003",
    "virtual": false
   },
   "O0003P0004-893751893750806906": {
    "id": "O0003P0004-893751893750806906",
    "label": "John",
    "type": "O0003",
    "virtual": false
   },
   "O0004P0002-HB1163": {
```

```
      "id": "O0004P0002-HB1163",
      "label": "HB1163",
      "type": "O0004",
      "virtual": false
    }
  },
  "nodesProps": {
    "O0003P0004-371411371416495795": {
      "O0003P0001":"Jane",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"371411371416495795"
    },
    "O0003P0004-893751893750806906": {
      "O0003P0001":"John",
      "O0003P0002":"",
      "O0003P0003":"",
      "O0003P0004":"893751893750806906"
    },
    "O0004P0002-HB1163": {
      "O0003P0001":"",
      "O0003P0002":"HB1163",
      "O0003P0003":""
    }
  }
 },
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```


6.6. Data persistence

6.6.1. PersistNode

You can call this operation to persist a node that does not exist in the system database to the database.

Request parameters

Name	Type	Required	Description
objectType	String	Yes	The object type. It is consistent with the configuration in Administration Console.
objectValue	String	Yes	Node value, a combination of the attribute label and the key attribute value.
properties	Array<Properties>	Yes	The property that needs to be added by the object.
propId	string	Yes	The object property ID. It is consistent with the object property configured in Administration Console.
propValue	string	Yes	The value corresponding to the object property.

Response parameters

Name	Type	Description
nodeCnt	Integer	The number of object nodes on a network.
nodes	Array<Node>	The node list.
id	String	The ID of the node.
label	String	The node label. It is consistent with the object property configured in Administration Console.
type	String	The node type. For example, O0001.
virtual	Boolean	Whether the node exists on the network.
nodesProps	Array<Property>	The node property list.
<Property>	<String,String>	The KV value of the node property. K indicates the node type, and it is consistent with the object type configured in Administration Console. V indicates the value of the node property, for example, "O0003P0001":"John".

Sample requests

```
{
  "objectType":"O0001",
  "objectValue":"O0001P0001-1323432543",
  "properties":[
    {
      "propId":"O0001P001",
      "propValue":"1323432543"
    },
    {
      "propId":"O0001P002",
      "propValue":"Alex"
    },
    {
      "propId":"O0001P003",
      "propValue":"Type C"
    },
    {
      "propId":"O0001P006",
      "propValue":"China Unicom"
    },
    {
      "propId":"O0001P007",
      "propValue":"1340005435"
    }
  ]
}
```

Sample responses

```
{
 "data": {
  "nodeCnt": 1,
  "nodes": {
   " O0001P0001-1323432543": {
    "id": " O0001P0001-1323432543",
    "label": "Alex",
    "avatar": "/static/img/id.png",
    "type": "O0001",
    "virtual": false
   }
  },
  "nodesProps": {
   " O0001P0001-1323432543": {
    " O0001P0001":"1323432543",
    " O0001P002":"Alex",
    " O0001P003":"Type C",
    " O0001P006":"China Unicom",
      " O0001P007":"2017-01-01 00:00:01"
   }
  }
 },
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```


6.6.2. PersistLink

You can call this operation to persist a link that does not exist in the system database to the database.

Request parameters

Name	Type	Required	Description
linkType	String,Array<String>	Yes	The list of start nodes in the KV structure. K indicates the object type, and it is consistent with the configuration in Administration Console. V indicates the object ID array.
source	Object	Yes	One object of the link.

Name	Type	Required	Description
objectType	String	Yes	The object type.
objectValue	int	No	The object value.
target	Object	Yes	The other object of the link.
objectType	String	No	The object type.
objectValue	String	No	The object value.
properties	Array<Properties>	Yes	The property that needs to be added by the object.
propId	string	Yes	The ID of the object property. It is consistent with the configuration in Administration Console.
propValue	string	Yes	The value corresponding to the object property.

Response parameters

Name	Type	Description
linkCnt	Integer	The number of links on a network.
links	Array<Link>	The link list.
id	String	The ID of the link.
source	String	The source object ID of the link.
sourceType	String	The source object type of the link. For example, O0003.
target	String	The target object ID of the link.
targetType	String	The target object type of the link. For example, O0004.
linkDetails	Array<String>	The ID list of link details contained in a link.
linkDetails	Array<LinkDetail>	The list of link details.
label	String	The label of the link detail.
linkId	String	The ID of the link detail.
linkType	String	The type of the link detail.
linkProps	Array<Property>	The property list of the link detail.

Name	Type	Description
Property	<String,String>	The KV value of the link property. K indicates the link type, and it is consistent with the link type configured in Administration Console. V indicates the value of the link property, for example, "L0003P0001":"Departure time".

Sample requests

```
{
  "linkType":"L00001",
  "source":{
    "objectType":"O0001",
    "objectValue":"O0001P001-13100000003"
  },
  "target":{
    "objectType":"O0001",
    "objectValue":"O0001P001-13100000004"
  },
  "properties":[
    {
      "propId":"L00001P013",
      "propValue":"13100000003"
    },
    {
      "propId":"L00001P015",
      "propValue":"13100000004"
    },
    {
      "propId":"L00001P012",
      "propValue":"14204320000"
    },
    {
      "propId":"L00001P022",
      "propValue":"1"
    },
    {
      "propId":"L00001P023",
      "propValue":"20"
    }
  ]
}
```

Sample responses

```
{
 "data": {
  "linkCnt": 1,
  "linkDetails": {
   "L00001^27b44b00fcf663b8fdfad968f8a2eba9": {
    "label": "Call",
    "linkId": "L0001^27b44b00fcf663b8fdfad968f8a2eba9",
    "linkType": "L0001",
    "d": "1",
   }
  },
  "linkProps": {
   "L0001^27b44b00fcf663b8fdfad968f8a2eba9": {
    "L0001P0001":"",
    "L0001P0002":"",
   }
  },
  "links": [
   {
    "id": "O0001#O0001P0001-13100000003^O0001#O0001P0001-13100000004",
    "linkDetails": [
     "L0001^27b44b00fcf663b8fdfad968f8a2eba9"
    ],
    "source": "O0001P0001-13100000003",
    "sourceType": "O0001",
    "target": "O0001P0001-13100000004",
    "targetType": "O0001"
   }
  ]
  }
 },
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```


6.7. OLEP escape

6.7.1. QueryOlpMetaConfig

You can call this operation to view the configurations of the OLEP model in Graph Analytics. Users from other systems can call this operation to configure service variable mappings.

Request parameters

None.

Response parameters

Name	Type	Description
objects	Array<ObjectDef>	All object definitions.
objDefId	String	The ID of the object definition. For example, O0001.
objName	String	The name of the object.
bizVar	String	The business variable.
objProps	Array<propertyDef>	The property list.
propDefId	String	The ID of the property definition. For example, O0001P001.
propName	String	The property name in Chinese.
bizVar	String	The business variable of the property.
links	Array<LinkDef>	All link definitions.
linkDefId	String	The ID of the object definition. For example, O0001.
linkName	String	The name of the object.
bizVar	String	The business variable.
linkProps	Array<propertyDef>	The property list.
propDefId	String	The ID of the property definition. For example, L0001P001.
propName	String	The property name in Chinese.
bizVar	String	The business variable of the property.

Sample requests

None.

Sample responses

```
{
      "data": {
  "objects":[
    {
      "objDefId":"O0001",
      "objName":"Mobile phone",
      "bizVar":"",
      "objProps":[
        {
          "propDefId":"O0001P001",
          "propName":"Mobile phone number",
          "bizVar":""
        },{
          "propDefId":"O0001P002",
          "propName":"Operator",
          "bizVar":""
        }
      ]
    },
  ],
  "links":[
    {
      "linkDefId":"L00001",
      "linkName":"Call",
      "bizVar":"tell",
      "linkProps":[
        {
          "propDefId":"L0001P001",
          "propName":"Mobile phone number",
          "bizVar":"phone"
        }
      ]
    }
  ]
      },
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```

6.7.2. UpdateOlpMetaConfig

You can call this operation to update the configuration of the Graph Analytics OLEP model. Users from other systems can call this operation to configure business variable mappings.

Request parameters

Name	Type	Description
objects	Array<ObjectDef>	All object definitions.
objDefId	String	The ID of the object definition. For example, O0001.
objName	String	The name of the object.
bizVar	String	The business variable.
objProps	Array<propertyDef>	The property list.
propDefId	String	The ID of the property definition. For example, O0001P001.
propName	String	The property name in Chinese.
bizVar	String	The business variable of the property.
links	Array<LinkDef>	All link definitions.
linkDefId	String	The ID of the object definition. For example, O0001.
linkName	String	The name of the object.
bizVar	String	The business variable.
linkProps	Array<propertyDef>	This indicates the property list.
propDefId	String	The ID of the property definition. For example, L0001P001.
propName	String	The property name in Chinese.
bizVar	String	The business variable of the property.

Response parameters

None.

Sample requests

```
{
  "objects":[
    {
      "objDefId":"O0001",
      "objName":"Mobile phone",
      "bizVar":"",
      "objProps":[
        {
          "propDefId":"O0001P001",
          "propName":"Mobile phone number",
          "bizVar":""
        },{
          "propDefId":"O0001P002",
          "propName":"Operator",
          "bizVar":""
        }
      ]
    },
  ],
  "links":[
    {
      "linkDefId":"L00001",
      "linkName":"Call",
      "bizVar":"tell",
      "linkProps":[
        {
          "propDefId":"L0001P001",
          "propName":"Mobile phone number",
          "bizVar":"phone"
        }
      ]
    }
  ]
}
```

Sample responses

```
{
      "data": "OK"
 "elapsedTime": 0,
 "noteMsg": "",
 "success": true
}
```

6.8. ShareAnalysis

You can call this operation to share your analysis cases with specific users and analyze these cases with them.

Request parameters

Name	Type	Description
ids	String,Array<String>	The ID list of the analysis files. For example, 6DA1362785C74F1CA8CE43A66F7F8FC9.
shareUsers	Array< ShareAnalysisUserGroupInfoBO>	Users who have the permission to view shared analysis files.
id	String	The ID of the user.
name	String	The name of the user.

Response parameters

None.

Sample requests

```
{
 "ids": [
  "6DA1362785C74F1CA8CE43A66F7F8FC9"
 ],
 "shareUsers": [
  {
   "id": 20,
   "name": "xianyi/xianyi"
  },
  {
   "id": 86,
   "name": "tiancheng"/tiancheng"
  }
 ]
}
```

Sample responses

```
{
 success: true,
 "errorcode": 0,
 noteMsg: "",
 data: null,
 elapsedTime: 391
}
```

6.9. Labels

6.9.1. AddLabels

You can call this operation to add label information to objects or events, so you can locate and query an object or event with ease.

Request parameters

Name	Type	Description
ids	Array<objectEventId>	The ID list of the object or event.
id	String	The ID of the object or event.

Name	Type	Description
type	String	The type of the node. A node can be an object or event.
label	String	The name of the label.
fill	String	The color of the label.
priority	int	The priority of the label.
analysisId	String	The ID of the analysis.
shareRange	int	The users to share the analysis with.
shareUsers	Array<ShareAnalysisUserGroupInfo>	The users who receive the analysis.
id	String	The ID of the user who receives the analysis.
name	String	The name of the user who receives the analysis.

Response parameters

None.

Sample requests

```
{
  "ids": [{
      "id": "O0001P0009-13888888888",
      "type": "O0001",
    },
    {
      "id": "O0001P0009-13888888889",
      "type": "O0001",
    }
  ],
  "label": "Prime suspect",
  "fill": "#FF0000",
  "priority" : 0,
  "analysisId": "admin_share_888",
  "shareRange": 4,
  "shareUsers": [{
    "id": 98,
    "name": "test"
  }]
}
```

Sample responses

```
{
 "success": true
 "errorcode": 0,
 "noteMsg": "",
 "data": {
  "O0001P0009-13888888888": {
   "id": "40",
   "label": "Prime suspect",
   "createID": "ht",
   "createName": "ht",
   "createTime": 1522571493,
   "fill": "#FF6600",
   "IsFake": true,
   "priority" : 0,
   "cnt": 0,
   "type":"ram-user",
   "shareRange": 1
  },
  "O0001P0009-13888888889": {
   "id": "41",
   "label": "Prime suspect",
   "createID": "ht",
   "createName": "ht",
   "createTime": 1522571493,
   "fill": "#FF6600",
   "IsFake": true,
   "priority" : 0,
   "count" : 0,
   "type":"user",
   "shareRange": 1
  }
 },
 "elapsedTime": 81
}
```


6.9.2. ObtainObjectLabels

You can call this operation to query the label information of the object or event being analyzed.

Request parameters

Name	Type	Description
analysisId	String	The ID of the analysis.
ids	Array<String>	The ID list of the object or event.

Response parameters

None.

Sample requests

```
{
 "analysisId": "DFA3B43C016440DE9005BC2A3D27CC5A",
 "ids": ["O0001P0009-13888888888"]
}
```

Sample responses

```
{
  "success": true
  "errorcode": 0,
  "noteMsg": "",
  "data": {
    "O0001P0009-13888888888": [{
      "id": "40",
      "label": "Prime suspect",
      "createID": "ht",
      "createName": "ht",
      "createTime": 1522571493,
      "fill": "#FF6600",
      "hasLike": true,
      "priority": 0,
      "count" : 0,
      "type": "user",
      "shareRange": 1
    }]
  },
  "elapsedTime": 7
}
```

6.9.3. ObtainLabelHistory

You can call this operation to retrieve the label operation history, including create, like, and unlike operations.

Request parameters

Name	Type	Description
id	String	The ID of the object or event.
tagId	Array<String>	The ID list of the label.
label	String	The name of the label.
fill	String	The color of the label.
cnt	int	The number of likes.
hasLike	Boolean	This indicates whether an object or event has been liked.
objType	String	The type of the node. A node can be an object or event.
priority	int	The priority of the label.
analysisId	String	The ID of the analysis.
type	String	The type of the node. A node can be an object or event.
shareRange	int	The users to share the analysis with.

Response parameters

None.

Sample requests

```
{
  "shareRange": 1,
  "hasLike": false,
  "cnt": 0,
  "id": "O00000322P0002-HB3037",
  "label": "Visible label_Initial green",
  "fill": "#4283e5",
  "priority": 3,
  "type": "O00000322",
  "tagId": "Visible label_Initial green 0",
  "objType": "O00000322",
  "analysisId": "08EFA103B0E4456890AF3B77C5D43DEC"
}
```


Sample responses


```
{
  "success": true,
  "errorcode": 0,
  "noteMsg": "",
  "data": [{
    "userId": "fangyi",
    "userName": "fangyi",
    "action": "Unlike",
    "time": 1524677337
  }, {
    "userId": "fangyi",
    "userName": "fangyi",
    "Action": "Like",
    "time": 1524677335
  }],
  "elapsedTime": 14
}
```


6.9.4. LikeLabels

You can call this operation to like a label of an object or event.

Request parameters

Name	Type	Description
id	String	The ID of the object or event.
tagId	Array<String>	The label ID list.
label	String	The name of the label.
fill	String	The color of the label.
priority	int	The priority of the label.
analysisId	String	The ID of the analysis.
type	String	The node type. A node can be an object or event.
shareRange	int	The users to share the analysis with.

Response parameters

None.

Sample requests

```
{
  "id": "O00000026P0001-0",
  "tagId": "9990",
  "label": "999",
  "fill": "#62C72C",
  "priority": 2,
  "analysisId": "94195146CB3C4125BF782D4E5B199A22",
  "type": "O00000026",
  "shareRange": 1
}
```

Sample responses

```
{
  "success": true,
  "errorcode": 0,
  "noteMsg": "",
  "data": {
    "id": "9990",
    "label": "999",
    "fill": "#62C72C",
    "hasLike": true,
    "priority": 2,
    "cnt": 2,
    "type": "user",
    "shareRange": 1
  },
  "elapsedTime": 575
}
```


6.9.5. CancelLikes

You can call this operation to cancel a like on a label of an object or event.

Request parameters

Name	Type	Description
id	String	The object or event ID.
tagId	Array<String>	The ID list of the label.
label	String	The content of the label.
fill	String	The color of the label.
priority	int	The priority of the label.
analysisId	String	The ID of the analysis.
type	String	The type of the node. A node can be an object or event.
shareRange	int	The users to share the analysis with.

Response parameters

None.

Sample requests

```
{
 "id": "O0022P001-750721750724104504",
 "tagId": "Prime suspect 0",
 "label": "Prime suspect",
 "fill": "#FF6600",
 "priority": 0,
 "analysisId": "DFA3B43C016440DE9005BC2A3D27CC5A",
 "type": "O0001",
 "shareRange": 1
}
```


Sample responses


```
{
  "success": true,
  "errorcode": 0,
  "noteMsg": "",
  "data": {
    "id": "Prime suspect 0",
    "label": "Prime suspect",
    "fill": "#FF6600",
    "hasLike": true,
    "priority": 0,
    "cnt": 2,
    "type": "user",
    "shareRange": 1
  },
  "elapsedTime": 73
}
```


6.9.6. ModifyLabels

You can call this operation to modify the labels of an object or event.

Request parameters

Name	Type	Description
analysisId	String	The ID of the analysis.
id	Array<String>	The ID list of the object or event.

Name	Type	Description
objType	String	The type of the node. A node can be an object or event.
old	Array	The original parameter.
id	String	The ID of the object or event.
label	String	The name of the label.
hasLike	Boolean	This indicates whether the object or event has been liked.
priority	int	The priority of the label.
cnt	int	The number of likes.
type	String	The type of the label.
shareRange	int	The users to share the analysis with.
tagId	String	The ID of the label.
objType	String	The type of the node. A node be an object or event.
analysisId	String	The ID of the analysis.
update	Array	The new parameter.
label	String	The name of the label.
fill	String	The color of the label.
priority	int	The priority of the label.
shareRange	Array	The users to share the analysis with.

Response parameters

None.

Sample requests

```
{
  "analysisId": "9EDB2FBEAA4C4E56B6508775D52B0907",
  "id": "O00000026P0001-0",
  "objType": "O00000026",
  "old": {
    "id": "O00000026P0001-0",
    "label": "111",
    "fill": "#62C72C",
    "hasLike": true,
    "priority": 2,
    "cnt": 1,
    "type": "user",
    "shareRange": 1,
    "tagId": "1110",
    "objType": "O00000026",
    "analysisId": "9EDB2FBEAA4C4E56B6508775D52B0907"
  },
  "update": {
    "label": "111",
    "fill": "#F8BD4F",
    "priority": 1,
    "shareRange": 2,
    "shareUsers": []
  }
}
```


Sample responses

```
{
  "success": true,
  "errorcode": 0,
  "noteMsg": "",
  "data": {
    "id": "1110",
    "label": "111",
    "fill": "#F8BD4F",
    "hasLike": true,
    "priority": 1,
    "cnt": 1,
    "type": "user",
    "shareRange": 2,
    "createID": "fy",
    "createName": "fy",
    "createTime": 1529478926
  },
  "elapsedTime": 227
}
```


6.9.7. DeleteLabels

You can call this operation to delete the labels of an object or event.

Request parameters

Name	Type	Description
id	String	The ID of the analysis.
tagId	Array<String>	The ID list of the label.
label	String	The name of the label.
fill	String	The color of the label.
priority	int	The priority of the label.
analysisId	String	The ID of the analysis.
type	String	The type of the node. A node can be an object or event.
shareRange	int	The users to share the analysis with.

Response parameters

None.

Sample requests

```
{
  "id": "O00000026P0001-0",
  "tagId": "1110",
  "label": "111",
  "fill": "#F8BD4F",
  "priority": 1,
  "analysisId": "9EDB2FBEAA4C4E56B6508775D52B0907",
  "shareRange": 2,
  "type": "O00000026"
}
```

Sample responses

```
{
  "success": true,
  "errorcode": 0,
  "noteMsg": "",
  "data": "Label deleted successfully",
  "elapsedTime": 41
}
```

6.10. Pattern recognition

6.10.1. ObtainPatternsandDirectories

You can call this operation to obtain all the patterns and directories, including private patterns, public patterns, private directories, and public directories.

Request parameters

None.

Response parameters

None.

Sample requests

None.

Sample responses

```
{
 "success": true,
 "errorcode": 0,
 "noteMsg": "",
 "data": {
  "publicList": [{
   "id": 81,
   "label": "Default directory",
   "owner": "system",
   "type": 1,
   "isPublic": 1,
   "children": [{
    "id": 170,
    "label": "TEST2",
    "owner": "test",
    "type": 2,
    "isPublic": 1,
    "children": []
   }, {
    "id": 183,
    "label": "Empty pattern",
    "owner": "tetest,
    "type": 2,
    "isPublic": 1,
    "children": []
   },{
     "id": 82,
     "label": "Test_Do not delete",
     "owner": "test",
    "type": 1,
    "isPublic": 1,
    "children": [{
     "id": 162,
     "label": "MYSQL triple closed loops",
     "owner": "test",
     "type": 2,
     "isPublic": 1,
     "children": []
   }
}
```

6.10.2. CreateTasks

You can call this operation to create pattern tasks. Each pattern can contain multiple tasks, and the parameters of each task can be modified.

Request parameters

Name	Type	Description
parentId	long	The ID of the parent directory.
isPublic	int	Whether the task is public or private.
label	String	The name of the task.

Response parameters

None.

Sample requests

```
{
  "parentId": 3,
  "isPublic": 1,
  "label": "Test 11",
}
```

Sample responses

None.

6.10.3. QueryTaskStatus

You can call this operation to regularly query the running status of the task.

Request parameters

Name	Type	Description
id	long	The ID of the task.

Response parameters

None.

Sample requests

```
status.json? id=15
```

Sample responses

```
{
    "success": true,
  "noteMsg": "",
  "data": {
    "status":0,
    "reason":""
  }
}
```

6.10.4. ViewTaskResults

You can call this operation to query the results of the task. If the task ID is specified, records of the specified task are returned, otherwise, the latest records of the current user are returned.

Request parameters

Name	Type	Description
taskId	long	The ID of the task.
pageSize	int	The number of pages.
pageIndex	int	The page number.

Response parameters

None.

Sample requests

```
result.json? id=267&activeVector=XXX&pageSize=10&pageIndex=1
```

Sample responses

```
{
 "success": true,
 "noteMsg": "",
 "data": {"taskId":120,
     "executor":"tc
     "duration":120,//s
     "start":1234567891,
     "complete":1234567891,
     "total":100,
     "pageSize":10,
     "pageIndex":0,
     "vector": [{ "checked": true,
             "id": "tab1",
             "title": "Matched pattern result one"},
            { "checked": false,
             "id": "tab1",
             "title": "Matched pattern result two" }],
 "tableHeader":["vertexLabel1","vertexLabel2"],
 "tableData":[
     ["id1","id3"],
     ["id2","id4"] ]
    }
}
```

6.11. Terms

Term	Description
object	
property	
link	
objectType	This describes codes starting with the letter O that are configured in Administration Console. For example, O0001.
linkType	This describes codes starting with the letter L that are configured in Administration Console. For example, L0001.
propertyId/propId	The property IDs of objects and links. For example, O0001P001 and L0001P001.

Term	Description
propertyType	Enumeration: string_equel: exact condition match. string_like: fuzzy match. number: query for numbers. time and date: time format. dict_select: exact word match.
propertyList	
derivated	The encoding is unavailable. Property conditions assembled under a certain logic can use the default settings configured in Administration Console.
statPropList	The statistical property conditions calculated by using the Count and Sum functions.
degree	A direct link with an object is called a first-degree link. A link with one intermediate point is called a second-degree link, and a link with two intermediate points is called a third-degree link. This pattern is consistent.
direct/d	Data direction of a link. For example, if A calls B, the link direction is A > B.

7.Elasticsearch (on ECS)

7.1. List of operations by function

Elasticsearch provides a RESTful API, which allows you to use HTTP API operations to manage Elasticsearch clusters. You can perform operations such as addition, deletion, modification, search, and alias configuration.

For more information, see Elasticsearch RESTful API.

Elasticsearch Reference [7.2]

Single-document operations

- Index
- Get
- Delete
- Update

Multi-document operations

- Multi get
- Bulk
- Delete by query
- Update by query
- Reindex

Use a Java REST client to access clusters

Only access over HTTP or TCP is supported. We recommend that you use a Java REST client provided by open source Elasticsearch.

Use the Java API to access clusters

Elasticsearch provides a built-in client for Java users to access clusters. For more information about the Java API, see Java API.

Transport Client

Transport Client allows you to send requests to a remote cluster. It is not a part of a cluster. It only forwards requests to nodes in a cluster.

Transport Client uses the Elasticsearch transport protocol to communicate with clusters over port `9300`.

Nodes in a cluster also use port `9300` to communicate with each other. Before you group nodes into a cluster, you must enable this port.

> **Note**
>
> Your nodes and Java clients must use the same Elasticsearch version. This way, the nodes and Java clients can identify each other.

RESTful API over HTTP

All other programming languages can use a RESTful API to communicate with Elasticsearch over port `9200`. You can use your preferred web client or the cURL tool to communicate with Elasticsearch.

> **Note**
>
> Open source Elasticsearch provides clients for several languages, including Groovy, JavaScript, .NET, PHP, Perl, Python, and Ruby.
>
> The open source community provides a number of clients and plug-ins. You can query the clients and plug-ins from open source Elasticsearch documentation.

Structure of a curl request over HTTP

```
curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

- *VERB*: the HTTP method. Valid values: GET, POST, PUT, HEAD, and DELETE.
- *PROTOCOL*: the protocol. Valid values: http and https. Use https only when HTTPS is enabled for Elasticsearch.
- *HOST*: the hostname of a node in your Elasticsearch cluster. If you use a local node, set this parameter to localhost.
- *PORT*: the port over which the Elasticsearch HTTP service runs. Default value: 9200.
- *PATH*: the endpoint of the API, such as `_count`. If _count is specified, the system returns the number of documents in a cluster. PATH can contain multiple segments, such as `_cluster/stats` and `_nodes/stats/jvm`.
- *QUERY_STRING*: the optional request parameters. For example, the ?pretty parameter makes the JSON-formatted data returned by the request easier to understand.
- *BODY*: the JSON-formatted request body. This field is required only when the request requires a body.

Examples

Run the following command to obtain the number of documents in an Elasticsearch cluster:

```
curl -XGET 'http://localhost:9200/_count?pretty' -d '
        {
        "query": {
        "match_all": {}
        }
        }'
```

Response:

```
{
        "count" : 0,
        "_shards" : {
        "total" : 5,
        "successful" : 5,
        "failed" : 0
        }
        }
```

Use the curl -i command to display the HTTP header:

```
curl -i -XGET 'localhost:9200/'
```

Complete sample request:

```
curl -XGET 'localhost:9200/_count?pretty' -d '
        {
        "query": {
        "match_all": {}
        }
        }'
```

Shortened sample request:

```
GET /_count
        {
        "query": {
        "match_all": {}
        }
        }
```

7.2. RESTful API

7.2.1. Single-document operations

7.2.1.1. Index operation

The index operation adds or updates a JSON document in a specific index to make the document searchable.

In the following example, the JSON document is inserted into the twitter index. The document type is _doc, and the document ID is 1.

```
PUT twitter/_doc/1
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

Response:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "result" : "created"
}
```

The `_shards` header provides information about the replication process of the index operation.

- `total` : indicates the number of shards and replicas on which the index operation is executed.
- `successful` : indicates the number of shards and replicas on which the index operation succeeds.
- `failed` : an array that contains replication errors when the index operation fails on replicas.

If the value of `successful` is greater than or equal to 1, the index operation succeeds.

> **Note** If the index operation succeeds, replicas may not all be started. Only shards must be started by default. This behavior can be changed. In this case, the value of `total` is equal to the total number of shards that is specified based on `number_of_replicas` , and the value of `successful` is equal to the total number of started shards and replicas. If no failures occur, the value of `failed` is 0.

7.2.1.1.1. Automatic index creation

The index operation automatically creates a specific index if the index does not exist. For more information about how to manually create an index, see Create index API. The index operation also automatically creates a dynamic mapping for a specific type if the mapping does not exist. For more information about how to manually create a type mapping, see Put mapping API.

Mappings are dynamic and do not require schemas. New fields and objects are automatically added to the mapping definition of a specific type. For more information about mapping definitions, see Mapping.

You can set `action.auto_create_index` to `false` in the configuration files of all nodes to disable automatic index creation. You can set `index.mapper.dynamic` to `false` to disable automatic mapping creation for each index.

Automatic index creation can contain a schema-based whitelist or blacklist. For example, you can set `action.auto_create_index` to `+aaa*,-bbb*,+ccc*,-*`. The plus sign (+) indicates allowed and the hyphen (-) indicates disallowed.

7.2.1.1.2. Versioning

Each indexed document has a version number. All associated `version` numbers are returned in the response to the index request. When you specify the `version` parameter, the index operation can determine whether to enable optimistic concurrency control. This controls the version of the document on which the index operation is intended to be executed. Versioning can be applied to scenarios such as performing a transactional read-then-update operation. You can specify `version` for the original document. This ensures that no changes will occur during the read process. If you want to update the document during the read process, we recommend that you set the `preference` parameter to `_primary`. Example:

```
PUT twitter/_doc/1?version=2
{
  "message" : "elasticsearch now has versioning support, double cool!"
}
```

> **Note** Versioning is implemented in real time and cannot be affected by near-real-time search operations. If no version is provided, the index operation is executed without version checks.

By default, internal versioning starts from 1 and increases with each update, including deletion. The version number can be supplemented with an external value in some cases, for example, the version number is maintained in a database. To enable this feature, set `version_type` to `external`. The provided value must be a long number that is greater than or equal to 0 and less than 9.2e + 18. When an external version type is used, the system does not check for a matching version number. Instead, it checks whether the version number that is transmitted to the index request is greater than the version number of the currently stored document. If the check result is true, the system indexes the document and uses the new version number for the document. If the provided value is less than or equal to the version number of the stored document, a version conflict occurs and the index operation fails.

> **Warning** External versioning considers 0 as a valid version number. This allows the version number to be synchronized with an external versioning system where version numbers start from 0 rather than 1. The disadvantage is that documents whose version numbers are 0 cannot be updated by calling the update by query operation or be deleted by calling the delete by query operation.

However, if version numbers from a source database are used, strict ordering is not required for asynchronized index operations that are executed as a result of changes to the source database. If external versioning is used, updating indexes by using data from a database is simplified. This is because only the latest version is used if the index operation fails.

Version types

In addition to the `internal` and `external` version types, Elasticsearch supports other version types for specific scenarios. The following content describes the supported version types and related semantics:

- `internal` : A document is indexed only when the provided version is the same as the version of the stored document.
- `external` or `external_gt` : A document is indexed only when the provided version is later than the version of the stored document or the document does not exist. The provided version is used as the new version and stored with the new document. The provided version must be a non-negative long number.
- `external_gte` : A document is indexed only when the provided version is later than or the same as the version of the stored document. If the document does not exist, the operation still succeeds. The provided version is used as the new version and stored with the new document. The provided version must be a non-negative long number.

> **Note** The `external_gte` version type is used for specific use cases. Proceed with caution. Improper use may cause data loss. The `force` version type is deprecated because it may cause inconsistency between shards and replicas.

7.2.1.1.3. Operation type

The index operation supports the `op_type` parameter to force a `create` operation, which allows the "put-if-absent" behavior. If the operation type is set to `create` and a document that has the same ID already exists in a specific index, the index operation fails.

Example:

```
PUT twitter/_doc/1?op_type=create
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

To set the operation type to `create`, you can also use the following URI:

```
PUT twitter/_doc/1/_create
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

7.2.1.1.4. Automatic ID generation

The index operation can be executed with no ID specified. In this case, the index operation automatically generates an ID. In addition, the `op_type` parameter is automatically set to `create`. Example (POST used instead of PUT):

```
POST twitter/_doc/
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

Response:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "W0tpsmIBdwcYyG50zbta",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "result": "created"
}
```

7.2.1.1.5. Routing

Shard placement or `routing` is controlled by using the hash of a document ID by default. To implement more precise control, you can use the `routing` parameter to specify the value that is fed into the hash function used by the router in each operation. Example:

```
POST twitter/_doc?routing=kimchy
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

In the preceding example, the _doc document is routed to a shard named kimchy based on the specified `routing` parameter.

When you create explicit mappings, you can use the `_routing` field to direct the index operation to extract the routing value from the document. This process requires a short period to parse the document and transmit the data. If the `_routing` mapping is defined and set to `required`, but the routing value is not specified or extracted, the index operation fails.

7.2.1.1.6. Distribution

The index operation is directed to a shard based on its routing. For more information, see the "Routing" topic. Then, it is executed on the actual node that stores this shard. After the shard completes the operation, updates are distributed to appropriate replicas if required.

7.2.1.1.7. Wait for active shards

To improve the flexibility of data write to the system, you can configure the index operation to proceed after the number of replicas for active shards reaches the specified threshold. If the replicas of the required number are unavailable, the write operation must wait and retry until the replicas are started or a timeout occurs. The default value of `wait_for_active_shards` is 1. This indicates that the write operation can proceed only when shards are active. You can specify `index.write.wait_for_active_shards` to dynamically overwrite the default value in index settings. To change this behavior of a specific operation, you can specify the `wait_for_active_shards` parameter in the relevant request.

This parameter can be set to `all` or any positive integer. The maximum value of this parameter is the total number of replicas per shard in an index. The total number of replicas per shard is `number_of_replicas` plus one. If a negative number or a number greater than the total number of replicas per shard is specified, an error will occur.

For example, you have a cluster that contains three nodes: `A`, `B`, and `C`. You create an index and set `number of replicas` to 3. In this case, the total number of configured replicas per shard is four, which is one more than the number of nodes. If you call the index operation, the operation proceeds only when each shard is available by default. If `A` hosts shards, the index operation still proceeds even if `B` and `C` fail. If you set `wait_for_active_shards` to `3` in the request and all the three nodes are running as expected, the index operation can proceed only when three replicas of each shard are active. This is because three nodes are active and each node must store one replica. However, if you set `wait_for_active_shards` to `all` or `4`, the index operation will not proceed because not all four replicas of each shard are active in the index. The operation will time out unless a new node is added to the cluster to host the fourth replica of each shard.

Note that this setting only reduces the possibility that the write operation does not write the required number of replicas. It cannot avoid this scenario because checks for the number of replicas are performed only before the write operation starts. After the write operation starts, replication may fail on some replicas but succeeds on shards. The number of replicas on which replication succeeds or fails are displayed in the `_shards` part of the response for the write operation.

```
{
  "_shards":{
    "total":2,
    "failed":0,
    "successful":2
  }
}
```

7.2.1.1.8. Refresh

This parameter determines when the changes made by a request are searchable. For more information, see Refresh.

7.2.1.1.9. Noop updates

When you call the index operation to update a document, a new version of the document is always created even if you do not update the document. If you want to enable the detection of noop updates, you can call the `_update` operation and set `detect_noop` to true. Noop updates are unavailable for the index operation because the index operation does not obtain old source code and cannot compare it with new source code.

You can enable or disable noop updates as needed. When you perform this operation, you must take several factors into account. The factors include the frequency at which the data source sends updates that are actually noops and the number of queries per second on the shard that receives the updates.

7.2.1.1.10. Timeout

When you call the index operation, the shard that is assigned to execute the index operation may be unavailable. The possible reasons are that the shard is recovering from a gateway or being reallocated. By default, the index operation waits for a maximum of one minute on the shard. If the shard is still unavailable after one minute, the index operation fails and an error is returned. The `timeout` parameter can be used to specify how long the operation waits. In the following example, the timeout parameter is set to five minutes.

```
PUT twitter/_doc/1?timeout=5m
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

7.2.1.2. Get operation

This topic describes the get operation.

You can call the get operation to obtain a JSON document from an index based on the document ID. In the following example, a JSON document is obtained from the twitter index. The document type is _doc, and the document ID is 0.

```
GET twitter/_doc/0
```

Response:

```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "0",
  "_version" : 1,
  "found": true,
  "_source" : {
    "user" : "kimchy",
    "date" : "2009-11-15T14:12:12",
    "likes": 0,
    "message" : "trying out Elasticsearch"
  }
}
```

The response contains the `_index` , `_type` , `_id` , and `_version` information of the document that you want to obtain. If the document is found, the value of the `found` field is true, and the `_source` field contains the content of the document.

The get operation also allows you to use `HEAD` to check whether a document exists. Example:

```
HEAD twitter/_doc/0
```

7.2.1.2.1. Real-time Get operation

The get operation is implemented in real time by default. It is not affected by the index refresh rate if data is searchable. If a document is updated but not refreshed, the get operation issues a refresh call to make the document searchable. This will also cause other documents that have been modified since the last refresh call to become searchable. To disable real-time get operations, you can set the `realtime` parameter to `false` .

7.2.1.2.2. Source filtering

By default, the get operation returns the content of the `_source` field unless the `stored_fields` parameter is specified or the `_source` field is disabled. You can run the following command to disable `_source` retrieval:

```
GET twitter/_doc/0?_source=false
```

The `_source` parameter is set to false.

If you need only one or two fields in `_source` , you can use the `_source_include` or `_source_exclude` parameter to include or exclude fields as needed. The two parameters are useful for partial retrieval of large documents because the parameters can reduce network overheads. Both parameters use a list of fields that are separated with commas (,) or use an expression that contains wildcards. Example:

```
GET twitter/_doc/0?_source_include=*.id&_source_exclude=entities
```

If you want to specify only the _source_include parameter, you can use a shorter notation. Example:

```
GET twitter/_doc/0?_source=*.id,retweeted
```

7.2.1.2.3. Stored fields

You can use the `stored_fields` parameter to specify a set of stored fields that you want to obtain. Requested fields that are not stored are ignored. For example, you can consider the following mapping:

```
PUT twitter
{
  "mappings": {
    "_doc": {
      "properties": {
        "counter": {
          "type": "integer",
          "store": false
        },
        "tags": {
          "type": "keyword",
          "store": true
        }
      }
    }
  }
}
```

Run the following command to add a document:

```
PUT twitter/_doc/1
{
  "counter" : 1,
  "tags" : ["red"]
}
```

Run the following command to retrieve the added document:

```
GET twitter/_doc/1?stored_fields=tags,counter
```

Response:

```
{
  "_index": "twitter",
  "_type": "_doc",
  "_id": "1",
  "_version": 1,
  "found": true,
  "fields": {
    "tags": [
      "red"
    ]
  }
}
```

Field values that are obtained from the document are always returned as an array. The `counter` field is not stored. Therefore, when the get operation attempts to obtain `stored_fields`, it ignores the counter field.

You can also retrieve metadata fields such as `_routing`. Example:

```
PUT twitter/_doc/2?routing=user1
{
  "counter" : 1,
  "tags" : ["white"]
  }
```

```
GET twitter/_doc/2?routing=user1&stored_fields=tags,counter
```

Response:

```
{
  "_index": "twitter",
  "_type": "_doc",
  "_id": "2",
  "_version": 1,
  "_routing": "user1",
  "found": true,
  "fields": {
    "tags": [
      "white"
    ]
  }
}
```

Only leaf fields can be returned by using the `stored_field` option. Object fields cannot be returned and such requests will fail.

7.2.1.2.4. Obtain _source

You can use the endpoint `/{index}/{type}/{id}/_source` to obtain only the `_source` field of a document. Example:

```
GET twitter/_doc/1/_source
```

You can also use source filtering parameters to specify the parts of `_source` to return. Example:

```
GET twitter/_doc/1/_source?_source_include=*.id&_source_exclude=entities'
```

You can use HEAD with the endpoint for _source to check whether the _source field exists. Example:

```
HEAD twitter/_doc/1/_source
```

If the _source field is disabled in the mappings setting, the document does not contain this field.

7.2.1.2.5. Routing

If the routing feature is used during indexing, you must specify the routing value to obtain a document. Example:

```
GET twitter/_doc/2?routing=user1
```

The preceding request obtains a document whose ID is `2` from the twitter index. However, the request will be routed based on the user indicated by routing. Note that the document cannot be obtained if you specify an invalid value for routing.

7.2.1.2.6. Preference

The `preference` parameter specifies the shard or replica on which the get operation is executed. By default, the operation is randomly executed among all shards and replicas.

Valid values for the `preference` parameter:

- `_primary` : The operation will be executed only on shards.
- `_local` : The operation will be executed on a locally allocated shard if required.

Custom (string) value

A custom value ensures that the same shards are used for this custom value. If different shards in different refresh states are available, only shards with the same custom value are used. The custom value can be a web session ID or a username.

7.2.1.2.7. Refresh

You can set the `refresh` parameter to `true` to refresh the related shard before the get operation is executed and make the shard searchable. Before you set it to `true` , verify that the setting does not cause system overload but slows down indexing.

7.2.1.2.8. Distribution

The get operation is hashed into a specific shard ID. It is then redirected to one of the replicas within the shard ID and returns a result. The replicas are the shard and its replicas within that shard ID. The more replicas, the better scaling performance of the get operation.

7.2.1.2.9. Versioning support

You can use the `version` parameter to retrieve a document only when the version of the document is the same as the specified version. This applies to all version types, except `FORCE` . Documents of this version type are always retrieved. The `FORCE` version type is already deprecated.

Elasticsearch has marked existing documents as deleted and added new documents. The documents that are marked as deleted still exist, but you cannot access them. If you continue to index data, Elasticsearch removes the documents in the background.

7.2.1.3. Delete operation

You can call the delete operation to delete a JSON document from a specific index based on the document ID. In the following example, a JSON document is deleted from the `twitter` index. The document type is `_doc` , and the document ID is `1` .

```
DELETE /twitter/_doc/1
```

Response:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_primary_term": 1,
  "_seq_no": 5,
  "result": "deleted"
}
```

7.2.1.3.1. Versioning

Each indexed document is versioned. When you delete a document, you can specify the `version` parameter to ensure that the document that you want to delete is actually being deleted and it is not modified during the deletion. Each write operation (including the deletion operation) executed on a document causes its version to be incremented. The version number of a deleted document remains available for a short period after deletion to allow control of concurrent operations. The duration that is required for the version of a deleted document to remain available depends on the setting of `index.gc_deletes`. The default value is 60 seconds.

7.2.1.3.2. Routing

If the routing feature is used during indexing, you must specify the routing value to delete a document. Example:

```
DELETE /twitter/_doc/1?routing=kimchy
```

The preceding request deletes a document whose ID is `1` from the twitter index. However, the request will be routed based on the user indicated by routing. Note that the document cannot be deleted if you specify an invalid value for routing.

If the `_routing` mapping is set to `required` but no routing value is specified, the delete operation will return `RoutingMissingException` and reject the request.

7.2.1.3.3. Automatic index creation

When an external versioning variant is used, the delete operation automatically creates a specific index if the index does not exist. For more information about how to manually create an index, see Create index API. The delete operation also automatically creates a dynamic mapping for a specific type if the mapping does not exist. For more information about how to manually create a type mapping, see Put mapping API.

7.2.1.3.4. Distribution

The delete operation is hashed into a specific shard ID. It is then redirected to a shard within that ID group and replicated (if required) to the replica of the shard within that ID group.

7.2.1.3.5. Wait for active shards

When you call the delete operation, you can specify the `wait_for_active_shards` parameter. This ensures that the delete operation is processed after the number of replicas for active shards reaches the minimum number. For more information, see Index API.

7.2.1.3.6. Refresh

This parameter specifies when the changes made by a request are refreshed and become searchable. For more information, see *?refresh* .

7.2.1.3.7. Timeout

When you call the delete operation, the shard that is assigned to perform the delete operation may be unavailable. The possible reasons are that the shard is recovering from a storage medium or being reallocated. The delete operation waits for a maximum of one minute on the shard by default. If the shard is still unavailable after one minute, the delete operation fails and an error is returned. The `timeout` parameter can be used to specify how long the operation waits. In the following example, the timeout parameter is set to five minutes.

```
DELETE /twitter/_doc/1?timeout=5m
```

7.2.1.4. Update operation

You can call the update operation to update a document based on a provided script. The operation obtains the document (collocated with the shard) from the index, runs the script (with optional script language and parameters), and reindexes the result (this operation can be deleted or ignored). It uses versioning to ensure that no updates have happened during the Get and Reindex operations.

Note: The update operation reindexes the entire document. It removes some network roundtrips and reduces chances of version conflicts between the Get and Index operations. You must enable the `_source` field to ensure that this operation works properly.

The following code provides an example of indexing a simple document:

```
PUT test/_doc/1
{
  "counter" : 1,
  "tags" : ["red"]
}
```

7.2.1.4.1. Updates by using scripts

To increase the counter, run the following command with a script specified:

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "ctx._source.counter += params.count",
    "lang": "painless",
    "params" : {
      "count" : 4
    }
  }
}
```

To add a tag to the tag list, run the following command with a script specified. If the tag already exists, it is still added.

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "ctx._source.tags.add(params.tag)",
    "lang": "painless",
    "params" : {
      "tag" : "blue"
    }
  }
}
```

In addition to the `_source` field, you can use the `ctx` mapping to obtain the following variables: `_index`, `_type`, `_id`, `_version`, `_routing`, and `_now`. The _now variable indicates the current timestamp.

You can add a field to the document. Example:

```
POST test/_doc/1/_update
{
  "script" : "ctx._source.new_field = 'value_of_new_field'"
}
```

You can also remove a field from the document. Example:

```
POST test/_doc/1/_update
{
  "script" : "ctx._source.remove('new_field')"
}
```

In addition, you can change the operation that is being executed. If the `tags` field contains `green`, the request deletes the document. Otherwise, the request performs `noop` operations.

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "if (ctx._source.tags.contains(params.tag)) { ctx.op = 'delete' } else { ctx.op = 'none' }",
    "lang": "painless",
    "params" : {
      "tag" : "green"
    }
  }
}
```

7.2.1.4.2. Update a part of a document

The update operation allows you to transmit a part of one document, which will be merged into an existing document. The following operations are included in the merging: simple recursive merging, inner merging of objects, and replacement of core key-value pairs and arrays. To replace the entire existing document, you must use the `index` operation instead. In the following example, a field is added to an existing document:

```
POST test/_doc/1/_update
{
  "doc" : {
    "name" : "new_name"
  }
}
```

If you specify both `doc` and `script`, `doc` is ignored. The best practice is to place the field pairs from a part of the document in script.

7.2.1.4.3. Detect noop updates

If you specify `doc`, its value is merged with the existing `_source`. By default, if the update operation detects an update with no changes, it returns "result": "noop". Example:

```
POST test/_doc/1/_update
{
  "doc" : {
    "name" : "new_name"
  }
}
```

If the value of `name` is `new_name` before the request is sent, the request is ignored. In this case, `noop` is returned in the `result` element of the response.

```
{
  "_shards": {
    "total": 0,
    "successful": 0,
    "failed": 0
  },
  "_index": "test",
  "_type": "_doc",
  "_id": "1",
  "_version": 6,
  "result": "noop"
}
```

If you want to disable the detection of noop updates, you can specify "detect_noop": false. Example:

```
POST test/_doc/1/_update
{
  "doc" : {
    "name" : "new_name"
  },
  "detect_noop": false
}
```

7.2.1.4.4. Upsert

If a document does not exist, the content of the `upsert` element is inserted as a new document. If the document exists, the following `script` is run:

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "ctx._source.counter += params.count",
    "lang": "painless",
    "params" : {
      "count" : 4
    }
  },
  "upsert" : {
    "counter" : 1
  }
}
```

scripted_upsert

If you want to run the script regardless of whether the document exists (the script instead of the `upsert` element initializes the document), set `scripted_upsert` to `true`.

```
POST sessions/session/dh3sgudg8gsrgl/_update
{
  "scripted_upsert":true,
  "script" : {
    "id": "my_web_session_summariser",
    "params" : {
      "pageViewEvent" : {
        "url":"foo.com/bar",
        "response":404,
        "time":"2014-01-01 12:32"
      }
    }
  },
  "upsert" : {}
}
```

doc_as_upsert

If you set `doc_as_upsert` to `true` instead of sending a part of `doc` and the `upsert` doc, the content of `doc` is used as the value of `upsert`. Example:

```
POST test/_doc/1/_update
{
  "doc": {
    "name": "new_name"
  },
  "doc_as_upsert": true
}
```

7.2.1.4.5. Parameters

The update operation supports the following parameters:

- `retry_on_conflict`

  Another process may have updated the same document in between the get and indexing phases of the update operation. In this case, the update operation fails due to a version conflict. The `retry_on_conflict` parameter specifies the number of retries before a version conflict is reported.

- `routing`

  This parameter is used to route an update request to the correct shard and sets routing for an upsert request when the document being updated does not exist. It cannot be used to update the routing of an earlier document.

- `timeout`

  The timeout period of waiting for a shard to become available.

- `wait_for_active_shards`

  The number of shard replicas that must be active before the update operation can proceed. For more information, see Index API.

- `refresh`

  This parameter specifies when the changes made by a request are refreshed and become searchable. For more information, see *?refresh*.

- `_source`

  This parameter specifies whether and how the updated source code is returned in the response. The updated source code is not returned by default. For more information, see Source filtering.

- `retry_on_conflict`

  The update operation uses the internal versioning of Elasticsearch to ensure that a document is unchanged during the update. You can specify the `version` parameter. This ensures that the document is updated only when its version is matched.

> **Note** The update operation supports only internal versioning.
>
> The update operation does not support external (version types `external` and `external_gte`) or forced (version type `force`) versioning because this will cause Elasticsearch version numbers to be inconsistent with the external system. You can use the `index` operation for external versioning.

7.2.2. Multi-document operations

7.2.2.1. Multi Get operation

The multi get operation allows you to obtain multiple documents based on an index, a type, an ID, and routing. The type and routing are optional. The response of the operation contains `docs`, which is an array of documents. In this array, the documents are sorted based on the order in which the information of the documents is specified in the multi get request. If a specific get operation fails, an object that contains this error is included in the response. The structure of the response for a successfully obtained document is the same as that in the response of the get operation.

Example:

```
GET /_mget
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1"
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2"
    }
  ]
}
```

The `mget` endpoint can be used based on an index. In this case, you do not need to specify the index in the request body. Example:

```
GET /test/_mget
{
  "docs" : [
    {
      "_type" : "_doc",
      "_id" : "1"
    },
    {
      "_type" : "_doc",
      "_id" : "2"
    }
  ]
}
```

The endpoint can also be used based on a type. Example:

```
GET /test/_doc/_mget
{
  "docs" : [
    {
      "_id" : "1"
    },
    {
      "_id" : "2"
    }
  ]
}
```

In this case, you can use the `ids` element to simplify the request. Example:

```
GET /test/_doc/_mget
{
  "ids" : ["1", "2"]
}
```

7.2.2.1.1. Source filtering

If the `_source` field is stored, it is returned for each document by default. Same as the get operation, you can use the `_source` parameter to retrieve only parts of `_source` or retrieve no data. You can also use the URL parameters `_source`, `_source_include`, and `_source_exclude` to specify default values for retrieval. If you do not specify retrieval conditions for a specific document, these default values are used.

Example:

```
GET /_mget
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "_source" : false
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2",
      "_source" : ["field3", "field4"]
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "3",
      "_source" : {
        "include": ["user"],
        "exclude": ["user.location"]
      }
    }
  ]
}
```

7.2.2.1.2. Fields

You can specify a storage field for each document to be retrieved, similar to the stored_fields parameter of the get operation. Example:

```
GET /_mget
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "stored_fields" : ["field1", "field2"]
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2",
      "stored_fields" : ["field3", "field4"]
    }
  ]
}
```

You can also specify the `stored_fields` parameter in a query string as a default value that is to be applied to all documents.

```
GET /test/_doc/_mget?stored_fields=field1,field2
{
  "docs" : [
    {
      "_id" : "1"
    },
    {
      "_id" : "2",
      "stored_fields" : ["field3", "field4"]
    }
  ]
}
```

1. Returns `field1` and `field2`.
2. Returns `field3` and `field4`.

7.2.2.1.3. Routing

You can specify the routing value as a parameter. Example:

```
GET /_mget?routing=key1
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "routing" : "key2"
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2"
    }
  ]
}
```

In this example, the document indicated by `test/_doc/2` will be obtained from the shard that corresponds to the `key1` routing key. The document indicated by `test/_doc/1` will be obtained from the shard that corresponds to the `key2` routing key.

7.2.2.1.4. Security

For more information, see *URL-based access control*.

7.2.2.2. Batch operation

You can call the bulk operation to index or delete multiple documents at a time. This operation significantly increases the indexing speed.

> **Note**
>
> The following clients that are officially supported provide helpers to assist in bulk request processing and reindexing of documents:
>
> - Perl: See Search::Elasticsearch::Client::5_0::Bulk and Search::Elasticsearch::Client::5_0::Scroll.
> - Python: See elasticsearch.helpers.

The endpoint of the RESTful API is `/_bulk`. It expects to return the following data in newline delimited JSON (NDJSON) format:

```
action_and_meta_data\n
optional_source\n
action_and_meta_data\n
optional_source\n
....
action_and_meta_data\n
optional_source\n
```

> **Note** The data in the last line must end with a newline character `\n`. Each newline character can be preceded by a carriage return `\r`. To send a request to the endpoint of the RESTful API, you must set the `Content-Type` header to `application/x-ndjson`.

You can perform the following operations: `indexing`, `creation`, `deletion`, and `update`. If you perform `indexing` or `creation` operations, make sure that a piece of source code is included in the next line. The two operations have the same semantics as the `op_type` parameter in the standard index API. If a document with the same index or type already exists, the creation operation fails and the indexing operation adds a document or replaces the existing document with a new one when necessary. The `deletion` operation does not require a piece of source code in the next line. It has the same semantics as the standard deletion API. The `update` operation requires that the doc, upsert, and script options be specified in the next line.

If you upload text files by using `curl` commands, you must use the `--data-binary` flag instead of the `-d` plaintext. This is because the latter does not retain newline characters. Example:

```
$ cat requests
{ "index" : { "_index" : "test", "_type" : "_doc", "_id" : "1" } }
{ "field1" : "value1" }
$ curl -s -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/_bulk --data-binary "@requests"; e
cho
{"took":7, "errors": false, "items":[{"index":{"_index":"test","_type":"_doc","_id":"1","_version":1,"result":"
created","forced_refresh":false}}]}
```

This format uses the combination of a backslash (\) and a lowercase letter n (`\n`) as delimiters. You must make sure that the JSON operations and source code are not prettified. The following code provides an example of a correct sequence of commands involved in a bulk operation:

```
POST _bulk
{"index":{"_index":"test","_type":"_doc","_id":"1"}}
{"field1":"value1"}
{"delete":{"_index":"test","_type":"_doc","_id":"2"}}
{"create":{"_index":"test","_type":"_doc","_id":"3"}}
{"field1":"value3"}
{"update":{"_id":"1","_type":"_doc","_index":"test"}}
{"doc":{"field2":"value2"}}
```

Sample response:

```
{
  "took": 30,
  "errors": false,
  "items": [
    {
      "index": {
        "_index": "test",
        "_type": "_doc",
        "_id": "1",
        "_version": 1,
        "result": "created",
        "_shards": {
          "total": 2,
          "successful": 1,
          "failed": 0
        },
        "status": 201,
        "_seq_no" : 0,
        "_primary_term": 1
      }
    },
    {
      "delete": {
        "_index": "test",
        "_type": "_doc",
        "_id": "2",
        "_version": 1,
        "result": "not_found",
        "_shards": {
          "total": 2,
```

```
      "successful": 1,
      "failed": 0
    },
    "status": 404,
    "_seq_no" : 1,
    "_primary_term" : 2
  }
},
{
  "create": {
    "_index": "test",
    "_type": "_doc",
    "_id": "3",
    "_version": 1,
    "result": "created",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "status": 201,
    "_seq_no" : 2,
    "_primary_term" : 3
  }
},
{
  "update": {
    "_index": "test",
    "_type": "_doc",
    "_id": "1",
    "_version": 2,
    "result": "updated",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "status": 200,
    "_seq_no" : 3,
    "_primary_term" : 4
```

```
    }
   }
  ]
}
```

The endpoints are `/_bulk`, `/{index}/_bulk`, and `{index}/{type}/_bulk`. If /{index}/_bulk or {index}/{type}/_bulk is provided, it will be used for bulk operations that do not explicitly specify the index or index/type unless otherwise instructed.

Note: Some operations are redirected to other shards on other nodes. Only `action_meta_data` is parsed on the receiver node to complete the processing as soon as possible.

Client libraries using this protocol should try to perform similar operations on the client, and reduce buffering as much as possible.

The response to a bulk operation involves large amounts of data in JSON format because the response contains the result of each operation that has been executed. The failure of a single operation does not affect the remaining operations.

The number of operations required in a single API call cannot be explicitly specified. You should attempt to use different settings to find the optimal one for your particular workload.

To call the HTTP operation, make sure that the client does not send HTTP chunks because this slows the processing speed.

Versioning

Each bulk item can include the version value indicated by the `version` field. It automatically follows the behavior of the indexing or deletion operation based on the `_version` mapping. It also supports `version_type`. For more information, see Versioning.

Routing

Each bulk item can include the routing value indicated by the `routing` field. It automatically follows the behavior of the indexing or deletion operation based on the `_routing` mapping.

Wait for active shards

To call a bulk operation, you can specify the `wait_for_active_shards` parameter to ensure that the bulk request is processed immediately after the active shard replicas reach the minimum. For more details and usage examples, see Index operation.

Refresh

You can specify this parameter to control the time when the changes made by a request are visible to search. For more information, see Refresh.

Update

When you perform the `update` operation, `retry_on_conflict` can be used as a field in the operation (not in the extra payload line), to specify how many times an update should be retried in the case of a version conflict.

The `update` operation payload supports the following options: `doc` (partial documents), `upsert`, `doc_as_upsert`, `script`, `params` (for scripts), `lang` (for scripts), and `_source`. For more information about the options, see the update documentation. Example of an update operation:

```
POST _bulk
{ "update" : {"_id" : "1", "_type" : "_doc", "_index" : "index1", "retry_on_conflict" : 3} }
{ "doc" : {"field" : "value"} }
{ "update" : { "_id" : "0", "_type" : "_doc", "_index" : "index1", "retry_on_conflict" : 3} }
{ "script" : { "source": "ctx._source.counter += params.param1", "lang" : "painless", "params" : {"param1" : 1}
}, "upsert" : {"counter" : 1}}
{ "update" : {"_id" : "2", "_type" : "_doc", "_index" : "index1", "retry_on_conflict" : 3} }
{ "doc" : {"field" : "value"}, "doc_as_upsert" : true }
{ "update" : {"_id" : "3", "_type" : "_doc", "_index" : "index1", "_source" : true} }
{ "doc" : {"field" : "value"} }
{ "update" : {"_id" : "4", "_type" : "_doc", "_index" : "index1"} }
{ "doc" : {"field" : "value"}, "_source": true}
```

Security

See *URL-based access control*.

7.2.2.3. Delete by query

In most cases, the `_delete_by_query` operation is used to delete each document that matches a query. Example:

```
POST twitter/_delete_by_query
{
 "query": {
  "match": {
   "message": "some message"
  }
 }
}
```

`"query"`: The query must be passed as a value to the `query` key in the same way as the query for the search operation. You can also use the `q` parameter in the same way as that for the search operation.

Sample response:

```
{
 "took" : 147,
 "timed_out": false,
 "deleted": 119,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "total": 119,
 "failures" : [ ]
}
```

The `_delete_by_query` operation obtains a snapshot of an index when the operation starts and deletes the document that is obtained by using `internal` versioning. If the document changes when the operation obtains the snapshot and deletes the document, a version conflict occurs. If versions match, the operation deletes the document.

> **Note** `Internal` versioning considers the value 0 as an invalid version number. Therefore, documents with this version number cannot be deleted by using the `_delete_by_query` operation. If you use this operation to delete such a document, the operation fails.

When `_delete_by_query` is executed, multiple search requests are executed in sequence to search for all matching documents that you want to delete. Each time documents are obtained, a bulk request is executed to delete the documents. If a search or bulk request is rejected, `_delete_by_query` uses exponential backoff to retry the request based on the default policy. A maximum of 10 retries are allowed. If the number of retries reaches the upper limit, `_delete_by_query` stops and returns all failures in the `failures` element of the response. The deletions that are performed remain valid. This indicates that the process is not rolled back. It is only stopped. Although the first failure causes the process to stop, all failures that are returned by the failed bulk request are included in the `failures` element. A large number of failures may be returned.

If you want to collect version conflicts but do not want to cause the process to stop, you can specify `conflicts=proceed` in the URL or `"conflicts": "proceed"` in the request body.

After you send a _delete_by_query request, tweets are deleted from the `twitter` index. Example:

```
POST twitter/_doc/_delete_by_query?conflicts=proceed
{
 "query": {
  "match_all": {}
 }
}
```

You can also delete documents of multiple indexes and multiple types at a time, in a similar way to the search operation. Example:

```
POST twitter,blog/_docs,post/_delete_by_query
{
 "query": {
  "match_all": {}
 }
}
```

If you specify `routing` in the request, it is replicated to a scroll query so that the process can be limited to the shards that match the routing value. Example:

```
POST twitter/_delete_by_query?routing=1
{
 "query": {
  "range" : {
   "age" : {
    "gte" : 10
   }
  }
 }
}
```

The default scroll batch size used by `_delete_by_query` is 1,000. You can use the `scroll_size` URL parameter to change the scroll batch size. Example:

```
POST twitter/_delete_by_query?scroll_size=5000
{
 "query":{
  "term":{
   "user":"kimchy"
  }
 }
}
```

7.2.2.3.1. URL parameters

In addition to standard parameters such as `pretty`, the _delete_by_query operation supports the following parameters: `refresh`, `wait_for_completion`, `wait_for_active_shards`, `timeout`, and `scroll`.

If the `refresh` parameter is specified, all the shards involved in the _delete_by_query request are refreshed immediately after the request is completed. This is different from the `refresh` parameter of the deletion operation. If the refresh parameter of the deletion operation is specified, only the shards that have received the deletion request are refreshed.

If the request contains `wait_for_completion=false`, Elasticsearch performs some prechecks, starts the request, and returns a task that can be used together with the task operation to cancel the task or query the status of the task. Elasticsearch also creates a record of this task as a document in `.tasks/task/${taskId}`. You can retain or delete this document as required. After you complete this operation, delete the document so that Elasticsearch can reclaim the disk space.

The `wait_for_active_shards` parameter controls how many replicas of a shard must be active before Elasticsearch proceeds with the request processing. For more information, see Index operation. The `timeout` parameter specifies the time period that a write request waits for unavailable shards to become available. The two operations work exactly the same way they work in the bulk operation. The `_delete_by_query` operation uses scrolled search. You can specify the `scroll` parameter to control how long it keeps the search context open, for example, `?scroll=10m`. The default value is 5m.

The `requests_per_second` parameter can be set to a positive decimal number, such as `1.4`, `6`, or `1000`. You can control the rate at which `_delete_by_query` issues batches of deletion operations by padding each batch with a wait time. You can set `requests_per_second` to `-1` to disable throttling.

A padding time between batches is used to throttle the rate so that the `_delete_by_query` operation can assign the internal scroll requests a timeout that takes the padding time into account. The padding time equals the difference between the batch size divided by `requests_per_second` and the time spent on writing. By default, the batch size is `1000`. If the `requests_per_second` parameter is set to `500`, the padding time is obtained based on the following calculation result:

```
target_time = 1000 / 500 per second = 2 seconds
wait_time = target_time - write_time = 2 seconds - .5 seconds = 1.5 seconds
```

The batch is issued as a single `_bulk` request. If the batch size is set to a large value, Elasticsearch has to create many requests and wait a period of time before proceeding with the next batch. This is "bursty" instead of "smooth". By default, requests_per_second is set to `-1`.

7.2.2.3.2. Response body

Sample response in JSON format:

```
{
 "took" : 147,
 "timed_out": false,
 "total": 119,
 "deleted": 119,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "failures" : [ ]
}
```

- `took`:

  The total number of milliseconds that the operation took.
- `timed_out`:

  If the requests executed during the _delete_by_query operation time out, this field is set to `true`.
- `total`:

  The number of documents that have been processed.
- `deleted`:

  The number of documents that have been deleted.
- `batches`:

The number of scroll responses revoked by the _delete_by_query operation.

- `version_conflicts` :

  The number of version conflicts caused by the _delete_by_query operation.

- `noops` :

  The value of this field is always zero in the response of the _delete_by_query operation. This field is reserved to ensure that the _delete_by_query, _update_by_query, and _reindex operations return responses in the same format.

- `retries` :

  The number of retries attempted by the _delete_by_query operation. `bulk` indicates the number of bulk operations retried and `search` indicates the number of search operations retried.

- `throttled_millis` :

  The number of milliseconds that the request was in the sleep state to conform to `requests_per_second` .

- `requests_per_second` :

  The number of requests per second that were executed during the _delete_by_query operation.

- `throttled_until_millis` :

  The value of this field is always zero in the response of the _delete_by_query operation. It is valid only when the delete by query operation is called. In this case, it indicates the time (in milliseconds since epoch) when the next throttling request will be executed to conform to `requests_per_second` .

- `failures` :

  An array of failures is returned if unrecoverable errors occur in this process. If this field is not empty, the request is stopped because of these failures. The _delete_by_query operation is implemented in batches. Any failures will cause the entire process to stop, but all failures in the current batch will be recorded in the array. You can use the `conflicts` option to prevent the _delete_by_query operation from stopping due to version conflicts.

7.2.2.3.3. Cancel tasks

You can call the task management operation to cancel a _delete_by_query tasks.

```
POST _tasks/task_id:1/_cancel
```

You can also call the task management operation to obtain `task_id` .

The cancellation may take several seconds. The task management operation continues to list the _delete_by_query task until this task checks that it has been canceled.

7.2.2.3.4. Task operation

You can call the task management operation to obtain the status of running _delete_by_query requests.

```
GET _tasks? detailed=true&actions=*/delete/byquery
```

Sample response:

```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
   "tasks" : {
    "r1A2WoRbTwKZ516z6NEs5A:36619" : {
     "node" : "r1A2WoRbTwKZ516z6NEs5A",
     "id" : 36619,
     "type" : "transport",
     "action" : "indices:data/write/delete/byquery",
     "status" : {
      "total" : 6154,
      "updated" : 0,
      "created" : 0,
      "deleted" : 3500,
      "batches" : 36,
      "version_conflicts" : 0,
      "noops" : 0,
      "retries": 0,
      "throttled_millis": 0
     },
     "description" : ""
    }
   }
  }
 }
}
```

`status` : This object contains the actual status. It is similar to the response in JSON format. The only difference is that the important `total` field is added to status. The `total` field indicates the total number of operations that the reindexing operation expects to perform. You can add the `updated` , `created` , and `deleted` fields to estimate the progress. When the sum of the values for the updated, created, and deleted fields is equal to the value of the `total` field, the request is terminated.

You can search for a task based on its ID.

```
GET /_tasks/taskId:1
```

The task operation has the following advantages: It integrates with `wait_for_completion=false` and returns the status of completed tasks in a transparent manner. If tasks are complete and `wait_for_completion=false` is specified, the operation returns a `results` or `error` field. Note that `wait_for_completion=false` will create a document in `tasks/task/${taskId}` . You can delete this document as needed.

7.2.2.3.5. Rethrottling

You can call the `_rethrottle` operation to change the value of `requests_per_second` for any running _delete_by_query task.

```
POST _delete_by_query/task_id:1/_rethrottle? requests_per_second=-1
```

You can also call the _rethrottle operation to obtain `task_id` .

You can set `requests_per_second` to `-1` to disable throttling in the same way as in the `_delete_by_query` operation. You can also set requests_per_second to any decimal number, such as `1.7` or `12` to control throttling. Rethrottling that speeds up a _delete_by_query task takes effect immediately. However, rethrottling that slows down a _delete_by_query task takes effect only after the current batch is processed. This prevents scroll timeouts.

7.2.2.3.6. Slicing

The _delete_by_query operation allows you to perform deletion in parallel with sliced scroll. During deletion, you can split a request into slices. This improves the efficiency of request processing.

Manual slicing

You can specify slice IDs and the total number of slices for each _delete_by_query request to manually split the request.

```
POST twitter/_delete_by_query
{
 "slice": {
  "id": 0,
  "max": 2
 },
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
POST twitter/_delete_by_query
{
 "slice": {
  "id": 1,
  "max": 2
 },
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

You can use the following method to check whether manual slicing is successful:

```
GET _refresh
        POST twitter/_search? size=0&filter_path=hits.total
{
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

A similar response with a reasonable `total` value is displayed:

```
{
 "hits": {
  "total": 0
 }
}
```

Automatic slicing

You can also call the sliced scroll operation to split `_uid`, so that a parallel _delete_by_query operation is automatically performed. The `slices` parameter is used to specify the number of slices to use.

```
POST twitter/_delete_by_query? refresh&slices=5
{
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

You can use the following method to check whether automatic slicing is successful:

```
POST twitter/_search? size=0&filter_path=hits.total
{
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

A similar response with a reasonable `total` value is displayed:

```
{
 "hits": {
  "total": 0
 }
}
```

If the `slices` parameter is set to `auto`, Elasticsearch determines the number of slices to use. In this case, each shard uses one slice until the number of slices reaches the upper limit. If multiple source indexes exist, Elasticsearch sets the number of slices based on the index that has the minimum number of shards.

If `slices` is added to `_delete_by_query`, the manual slicing process becomes automated. This method generates subrequests with different characteristics.

- You can view these subrequests in the task operation. These subrequests are subtasks of the request tasks that contain `slices`.
- When you query the status of request tasks that contain `slices`, only the status of the tasks that have completed slicing can be obtained.
- These subrequests can be separately processed for throttling cancellation and rethrottling operations.
- If you use `slices` to rethrottle a request, only unfinished subrequests are rethrottled proportionally.
- If you use `slices` to cancel a request, all its subrequests are canceled.
- When `slices` is used, the document segments are not evenly allocated to each subrequest. All documents will be processed, but some slices may be greater than other slices. Greater slices may have a more even distribution.
- In requests that contain `slices`, the `requests_per_second` and `size` parameters are allocated to each subrequest proportionally. Based on the uneven distribution principle, a request that contains `slices` and a specified `size` may not return a deleted document that has the specified `size` value.
- The source index snapshots obtained by each subrequest are slightly different, but these snapshots

are taken at approximately the same time.

Configure the number of slices

For automatic slicing where `slices` is set to `auto`, Elasticsearch sets a reasonable number of slices for most indexes. Use the following guidelines if you want to perform manual slicing or adjust automatic slicing:

If the value of `slices` is equal to the number of shards in an index, you can obtain the optimal query performance. A value greater than the number of shards in an index, for example, 500, may not improve efficiency but increase overheads, resulting in performance degradation.

Deletion performance scales linearly across available resources with the number of slices.

Both the documents that are being reindexed and cluster resources determine whether query or deletion performance is dominant during the runtime.

7.2.2.4. Update by query

The most basic usage of the `_update_by_query` operation is to perform updates on each document in an index. In this case, you do not need to change the source code. It is useful to obtain new properties or several other online mapping changes. Example:

```
POST twitter/_update_by_query? conflicts=proceed
```

The following or similar result is returned:

```
{
 "took" : 147,
 "timed_out": false,
 "updated": 120,
 "deleted": 0,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "total": 120,
 "failures" : [ ]
}
```

The `_update_by_query` operation obtains a snapshot of the index when the process of this operation starts and indexes the document that is found during `internal` versioning. Version conflicts may occur when a document changes. The document change occurs after the snapshot is obtained and before the index request is processed. If the versions match, the document is updated and the version number is incremented.

> **Note** The document with version number 0 cannot be updated by calling the `_update_by_query` operation. This is because `internal` versioning considers the value 0 as an invalid version number and the update requests from the document will fail.

All update and query failures cause the `_update_by_query` operation to stop and these failures are included in the `failures` element of the response. The _update_by_query operation that has been performed is still effective. This means that the process of this operation is not restarted but is stopped. Although the first failure causes the process to stop, all failures that are returned by the failed bulk request are included in the `failures` element. Therefore, a few failed entities may exist.

If you want to count only version conflicts without stopping the `_update_by_query` operation, set `conflicts=proceed` in the URL, or set `"conflicts": "proceed"` in the request body. In the preceding example of the _update_by_query operation, this setting is used because the `_update_by_query` operation only tries to obtain an online mapping change. Version conflicts may occur when a document changes. The document change occurs after the _update_by_query operation starts and before the change operations are performed. This is normal because the _update_by_query operation has obtained the online mapping change.

After the operation format is restored, tweets are updated from the `twitter` index.

```
POST twitter/_doc/_update_by_query? conflicts=proceed
```

You can also use Query DSL to query the `_update_by_query` operation. This way, all documents are updated from the `twitter` index for user `kimchy`.

```
POST twitter/_update_by_query? conflicts=proceed
{
 "query": {
  "term": {
   "user": "kimchy"
  }
 }
}
```

`"query"` must be passed as a value to the `query` key in the same way as the search operation. You can also use the `q` parameter in the same way as the search operation.

The preceding description demonstrates how to call the _update_by_query operation to update a document without the need to change the source code. This is a helpful operation for purposes such as obtaining new properties . You can also call the `_update_by_query` operation to update a document by using scripts. This will increment the `likes` field on all the tweets for user kimchy.

```
POST twitter/_update_by_query
{
 "script": {
  "source": "ctx._source.likes++",
  "lang": "painless"
 },
 "query": {
  "term": {
   "user": "kimchy"
  }
 }
}
```

Like the update operation, the _update_by_query operation allows you to specify `ctx.op` if you want to change the operation that is about to execute.

- `noop` : Set ctx.op to `noop` if your script determines that the document does not need to be changed. This way, the `_update_by_query` operation omits this documents from its updates. The `noop` counter reports "no operation" in the response body.
- `delete` : Set ctx.op to `delete` if your script determines that the document must be deleted. The `deleted` counter reports this deletion operation in the response body.

If you do not set `ctx.op` to noop or delete, an error is returned. The same rule applies to other fields in `ctx` .

Note that `conflicts=proceed` is not specified in this case. This is because we want to stop the process when a version conflict occurs.

The_update_by_query operation does not allow you to move the documents involved, but only allows you to modify their source code. No provisions are provided to remove the document from its original location.

You can also perform these operations on multiple indexes and multiple types at a time, in the same way as the search operation:

```
POST twitter,blog/_doc,post/_update_by_query
```

If you provide `routing` , the routing is copied to the scroll query, so that the process is limited to the shards that match the routing value.

```
POST twitter/_update_by_query? routing=1
```

By default, `_update_by_query` uses 1,000 scroll batches. You can use the `scroll_size` URL parameter to change the batch size.

```
POST twitter/_update_by_query? scroll_size=100
```

The `_update_by_query` operation also allows you to use an ingest node based on a specified `pipeline`, as shown in the following code:

```
PUT _ingest/pipeline/set-foo
{
 "description" : "sets foo",
 "processors" : [ {
   "set" : {
    "field": "foo",
    "value": "bar"
   }
 } ]
}
POST twitter/_update_by_query? pipeline=set-foo
```

7.2.2.4.1. URL parameters

In addition to the standard parameters such as `pretty`, the Update By Query operation supports the following parameters: `refresh`, `wait_for_completion`, `wait_for_active_shards`, `timeout`, and `scroll`.

If the `refresh` parameter is sent after a request is complete, all shards in the index that is being updated will be updated. This is different from the `refresh` parameter for the index operation. The latter one only indexes the shards that receive the new data.

If the request contains the `wait_for_completion=false` parameter, Elasticsearch runs some prechecks, initiates the request, and returns a task that can be used in combination with the task operation to cancel the task or query the status of the `task`. Elasticsearch can also create a record of this task as a document in `.tasks/task/${taskId}`. You can keep or delete the document as needed. After the task is complete, delete the document so that Elasticsearch can reclaim the disk space.

The `wait_for_active_shards` parameter controls how many replicas of a shard must be activated before Elasticsearch proceeds with the request. For more information, see Active shards. The `timeout` parameter controls how long each write request waits for unavailable shards to become available. Both parameters work exactly the same way they work inbulk operation. `_update_by_query` uses scroll search. You can set the `scroll` parameter to control how long it keeps the search context alive, for example, `? scroll=10m`. The default value is 5m.

The `requests_per_second` parameter can be set to a positive decimal number, for example, `1.4`, `6`, or `1000`. This parameter is used together with the wait time to throttle the rate at which `_update_by_query` issues batches of index operations. You can set `requests_per_second` to `-1` to disable throttling.

A padding time between batches is used to throttle the rate so that the `_update_by_query` operation can assign the internal scroll requests a timeout that takes the padding time into account. The padding time equals the difference between the batch size divided by `requests_per_second` and the time spent on writing. By default, the batch size is `1000`. If the `requests_per_second` parameter is set to `500`, the padding time is obtained based on the following calculation result:

```
target_time = 1000 / 500 per second = 2 seconds
wait_time = target_time - delete_time = 2 seconds - .5 seconds = 1.5 seconds
```

The batch is issued as a single `_bulk` request. Large batch sizes will cause Elasticsearch to create many requests and then wait for a while before starting the next batch. This is "bursty" instead of "smooth". By default, requests_per_second is set to `-1`.

7.2.2.4.2. Response body

Sample response in JSON format:

```
{
 "took" : 147,
 "timed_out": false,
 "total": 5,
 "updated": 5,
 "deleted": 0,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "failures" : [ ]
}
```

- `took`: the total number of milliseconds that the operation took.

- `timed_out` : If a request times out during the_update_by_query operation, this field is set to `true` .
- `Total` : the number of documents that have been processed.
- `updated` : the number of documents that have been updated.
- `deleted` : the number of documents that have been deleted.
- `batches` : the number of scroll responses revoked by the _update_by_query operation.
- `version_conflicts` : the number of version conflicts caused by the _update_by_query operation.
- `noops` : the number of documents that were omitted because the script used for the update by query returned a `noop` value for `ctx.op` .
- `retries` : the number of retries attempted by the update by query operation. `bulk` indicates the number of bulk operations retried. `search` indicates the number of search operations retried.
- `throttled_millis` : the number of milliseconds that the request is in sleep mode to conform to `reques ts_per_second` .
- `requests_per_second` : the number of requests per second that are effectively executed during the _update_by_query operation.
- `throttled_until_millis` : This field must be zero in a delete by query response. It is valid only when the task operation is called. In this situation, this field indicates that a throttling request will be executed again to conform to `requests_per_second` since epoch in milliseconds.
- `failures` : An array of failures is returned if unrecoverable errors occur in this process. If this field is not empty, the request is terminated because of these failures. The _update_by_query operation is implemented in batches. Any failures will cause the entire process to stop, but all failures in the current batch will be recorded in the array. You can use the `conflicts` option to prevent the _update_by_query operation from stopping due to version conflicts.

7.2.2.4.3. Task operation

You can call the task operation to obtain the status of running update-by-query requests.

```
GET _tasks? detailed=true&actions=*byquery
```

Sample response:

```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
   "tasks" : {
    "r1A2WoRbTwKZ516z6NEs5A:36619" : {
     "node" : "r1A2WoRbTwKZ516z6NEs5A",
     "id" : 36619,
     "type" : "transport",
     "action" : "indices:data/write/update/byquery",
     "status" : {
      "total" : 6154,
      "updated" : 3500,
      "created" : 0,
      "deleted" : 0,
      "batches" : 4,
      "version_conflicts" : 0,
      "noops" : 0,
      "retries": {
       "bulk": 0,
       "search": 0
      }
      "throttled_millis": 0
     },
     "description" : ""
    }
   }
  }
 }
}
```

"status" : The object contains the actual status. It is similar to the response in JSON format. The only difference is that the important total field is added to status. The total field indicates the total number of operations that the _update_by_query operation expects to perform. You can add the updated , created , and deleted fields to estimate the progress. The request will finish when the sum is equal to the value of the total field.

You can search for a task based on its ID:

```
GET /_tasks/taskId:1
```

The advantage of this API is that it integrates with the wait_for_completion=false to return the status of completed tasks in a transparent manner. If the task is completed and wait_for_completion=false is set for it, this task will return the results or error field. Note that wait_for_completion=false will create a document in .tasks/task/${taskId} . You can delete this document as needed.

7.2.2.4.4. Cancel tasks

You can call the task management operation to cancel an update by query task.

```
POST _tasks/task_id:1/_cancel
```

You can also call the task management operation to obtain task_id .

The cancellation may take several seconds. The task management operation continues to list the update by query task until this task checks that it has been canceled.

7.2.2.4.5. Rethrottling

You can call the _rethrottle operation to change the value of requests_per_second on a running update by query task.

```
POST _update_by_query/task_id:1/_rethrottle? requests_per_second=-1
```

You can also call the task management operation to obtain task_id .

You can set requests_per_second to -1 to disable throttling in the same way as in the _update_by_query operation. Alternatively, you can set this field to a decimal number such as 1.7 or 12 to control throttling. Rethrottling that speeds up an update by query task takes effect immediately. However, rethrottling that slows down an update by query task takes effect only after the current batch is processed. This prevents scroll timeouts.

7.2.2.4.6. Slicing

You can call the _update_by_query operation and the sliced scroll operation at the same time. Simultaneous operations allow you to split requests into smaller parts. This improves the efficiency of request processing.

Manual slicing

You can specify slice IDs and the total number of slices for each _update_by_query request to manually split the request.

```
POST twitter/_update_by_query
{
 "slice": {
  "id": 0,
  "max": 2
 },
 "script": {
  "source": "ctx._source['extra'] = 'test'"
 }
}
POST twitter/_update_by_query
{
 "slice": {
  "id": 1,
  "max": 2
 },
 "script": {
  "source": "ctx._source['extra'] = 'test'"
 }
}
```

You can use the following method to check whether manual slicing is successful:

```
GET _refresh
        POST twitter/_search? size=0&q=extra:test&filter_path=hits.total
```

A similar response with a suitable `total` value appears:

```
{
 "hits": {
  "total": 120
 }
}
```

Automatic slicing

You can also call the sliced scroll operation to split `_uid`, so that a parallel _update_by_query

operation is automatically performed. `slices` is used to specify the number of slices you use.

```
POST twitter/_update_by_query? refresh&slices=5
{
 "script": {
  "source": "ctx._source['extra'] = 'test'"
 }
}
```

You can use the following method to check whether automatic slicing is successful:

```
POST twitter/_search? size=0&q=extra:test&filter_path=hits.total
```

A similar response with a suitable `total` value appears:

```
{
 "hits": {
  "total": 120
 }
}
```

If the `slices` parameter is set to `auto`, Elasticsearch will determine the number of slices to use. In this case, each shard uses one slice until the number of slices reaches the upper limit. If multiple source indexes exist, Elasticsearch will set the number of slices based on the index that has a minimum number of shards.

If `slices` is added to `_update_by_query`, the manual slicing process becomes automated. This method generates subrequests with different characteristics.

- You can view these subrequests in the task operation. These subrequests are subtasks of the request tasks that contain `slices`.
- When you query the status of request tasks that contain `slices`, only the status of the tasks that have completed slicing can be obtained.
- These subrequests can be separately processed for throttling cancellation and rethrottling operations.
- If you use `slices` to rethrottle a request, only unfinished subrequests are proportionally rethrottled.
- If you use `slices` to cancel a request, all its subrequests are canceled.
- When `slices` is used, the document segments are not evenly allocated to each subrequest. All documents will be processed. However, some slices may be greater than other slices. Greater slices may have a more even distribution.
- In requests that contain `slices`, the `requests_per_second` and `size` parameters are proportionally allocated to each subrequest. Based on the uneven distribution principle, a request that contains `slices` and a specified `size` may not return an updated document that has the specified `size` value.

- The source index snapshots obtained by each subrequest are slightly different, but these snapshots are taken at approximately the same time.

Choose the number of slices

For automatic slicing where `slices` is set to `auto`, Elasticsearch sets a suitable number of slices for most indexes. If you want to use manual slicing or other methods to adjust the number of slices choosed for automatic slicing, take note of the following points:

If the value of `slices` is equal to the number of shards in an index, you can obtain the optimal query performance. A value greater than the number of shards in an index, for example, 500, may not improve efficiency but increase overheads, resulting in performance degradation.

Update performance scales linearly across available resources with the number of slices.

Both the documents that are being reindexed and cluster resources determine whether query or update performance is dominant during the runtime.

7.2.2.4.7. Add new properties

Assume that you have created an index without dynamic mappings. Then, you add data to the index and add a mapping value to obtain more fields from the data. Sample code:

```
PUT test
{
 "mappings": {
  "_doc": {
   "dynamic": false,
   "properties": {
    "text": {"type": "text"}
   }
  }
 }
}
POST test/_doc? refresh
{
 "text": "words words",
 "flag": "bar"
}
POST test/_doc? refresh
{
 "text": "words words",
 "flag": "foo"
}
PUT test/_mapping/_doc
{
 "properties": {
  "text": {"type": "text"},
  "flag": {"type": "text", "analyzer": "keyword"}
 }
}
```

`"dynamic"` : This field is set to false, indicating that the new field will not be indexed and will only be stored in `_source` .

`PUT test/_mapping/_doc` : This code indicates that the mapping will be updated to add a new `flag` field. To add a new field, reindex all documents by using this field.

If you search for data records by using this field, no results will be returned.

```
POST test/_search? filter_path=hits.total
{
 "query": {
  "match": {
   "flag": "foo"
  }
 }
}
```

```
{
 "hits" : {
  "total" : 0
 }
}
```

However, you can initiate a `_update_by_query` request to add the new mapping:

```
POST test/_update_by_query? refresh&conflicts=proceed
POST test/_search? filter_path=hits.total
{
 "query": {
  "match": {
   "flag": "foo"
  }
 }
}
```

```
{
 "hits" : {
  "total" : 1
 }
}
```

You can perform the same operation if you want to add a field to a multifield.

7.2.2.5. Reindex

> **Note** The reindex operation does not attempt to specify the destination index. It does not copy the settings of the source index. To perform the `_reindex` operation, you must specify information about the destination index, including mappings, shard counts, and replicas.

In its basic form, the `_reindex` operation copies a document from one index to another. In the following example, documents are copied from the `twitter` index to the `new_twitter` index:

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

Sample response:

```
{
 "took" : 147,
 "timed_out": false,
 "created": 120,
 "updated": 0,
 "deleted": 0,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "total": 120,
 "failures" : [ ]
}
```

Similar to `_update_by_query`, `_reindex` obtains a snapshot of the source index. To avoid version conflicts, the destination index that is created based on the snapshot must be different from the source index. The `dest` element can be configured in a similar way as that in the index operation. This allows you to implement optimistic concurrency control. If you omit the `version_type` parameter or set it to `internal`, Elasticsearch dumps documents to the destination index. This will overwrite any documents that have the same type and ID:

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "version_type": "internal"
 }
}
```

If you set `version_type` to `external`, Elasticsearch retains `version` in the source document, creates missing documents, and updates any documents whose version in the destination index is earlier than its version in the source index.

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "version_type": "external"
 }
}
```

If you set `op_type` to `create`, Elasticsearch calls the `_reindex` operation. This allows you to create the missing documents in the destination index. In this case, all existing documents will cause version conflicts.

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "op_type": "create"
 }
}
```

Version conflicts automatically stop a `_reindex` process. However, if you set `"conflicts": "proceed"`, version conflicts are only counted and do not stop the _reindex process.

```
POST _reindex
{
 "conflicts": "proceed",
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "op_type": "create"
 }
}
```

You can add a type or query for `source` to limit the number of documents. This will only copy tweets made from `kimchy` to `new_twitter`.

```
POST _reindex
{
 "source": {
  "index": "twitter",
  "type": "_doc",
  "query": {
   "term": {
    "user": "kimchy"
   }
  }
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

The `index` and `type` fields in `source` can be lists. This allows you to copy multiple documents from a large amount of source code by using one request. After you specify the two fields, you can copy documents of the `_doc` and `post` types from the `twitter` and `blog` indexes. The copied documents will include the `post` type in the `twitter` index and the `_doc` type in the `blog` index. To configure more parameters for the operation, you can specify parameters in the `query` field.

The reindex operation does not handle ID conflicts. When ID conflicts occur, the destination index still remains valid. However, the iteration order is not well defined. In this way, you cannot predict which document is retained.

```
POST _reindex
{
 "source": {
  "index": ["twitter", "blog"],
  "type": ["_doc", "post"]
 },
 "dest": {
  "index": "all_together"
 }
}
```

You can specify the `size` field to limit the number of documents you want to process. If you set size to 1, only one document will be copied from `twitter` to `new_twitter`:

```
POST _reindex
{
 "size": 1,
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

You can use `sort` to copy a specific set of documents from the `twitter` index. Sorting makes the scroll less efficient but it is useful in some scenarios. We recommend that you use the `size` or `sort` field. In the following example, 10,000 documents are copied from `twitter` to `new_twitter`:

```
POST _reindex
{
 "size": 10000,
 "source": {
  "index": "twitter",
  "sort": { "date": "desc" }
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

The `source` section supports all the elements that are supported in a search request. For example, a subset of the fields from the original documents can be reindexed by using `source` filtering, as shown in the following code:

```
POST _reindex
{
 "source": {
  "index": "twitter",
  "_source": ["user", "_doc"]
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

Like `_update_by_query`, `_reindex` supports a script that is used to modify documents. Unlike `_update_by_query`, the script is allowed to modify the metadata in a document. The following code is used to raise the version of the source document:

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "version_type": "external"
 },
 "script": {
  "source": "if (ctx._source.foo == 'bar') {ctx._version++; ctx._source.remove('foo')}",
  "lang": "painless"
 }
}
```

Similar to `_update_by_query`, the reindex operation allows you to set `ctx.op` to change the operation you want to perform on the destination index.

- `noop`:

  Set `ctx.op` to noop if the script field specifies that it is unnecessary to index the document in the destination index. This operation will be reported in the `noop` counter in the response body.

- `delete`:

  Set ctx.op to `delete` if the script field specifies that the document must be deleted from the destination index. The deletion operation will be reported in the `deleted` counter in the response body.

If you do not set `ctx.op` to noop or delete, an error will be returned. The same rule applies to other fields in `ctx`.

Use caution when you modify the following fields:

- `_id`
- `_type`
- `_index`
- `_version`
- `_routing`

If you set `_version` to `null` or clear this field from `ctx`, the version will not be included in an index request. This causes the document to be overwritten in the target index regardless of the target version or the version type you use in the `_reindex` request.

If `_reindex` discovers a document that contains routing, the routing will be retained unless it is changed by the script. In a `dest` request, you can set `routing` to the following values:

- `keep`

  Sets the routing in a bulk request sent for each match item to the routing for the match item. This is the default value.

- `discard`

  Sets the routing in a bulk request sent for each match item to `null`.

- `=<some text>`

  Sets the routing in a bulk request sent for each match item to all texts after `=`.

For example, you can use the following request to copy all documents from the `source` index with the company name `cat` to the `dest` index with routing set to `cat`:

```
POST _reindex
{
 "source": {
  "index": "source",
  "query": {
   "match": {
    "company": "cat"
   }
  }
 },
 "dest": {
  "index": "dest",
  "routing": "=cat"
 }
}
```

By default, `_reindex` uses 1,000 scroll batches. You can specify the `size` field in the `source` element to change the batch size.

```
POST _reindex
{
 "source": {
  "index": "source",
  "size": 100
 },
 "dest": {
  "index": "dest",
  "routing": "=cat"
 }
}
```

The reindex operation also allows you to use an ingest node by specifying `pipeline`, as shown in the following code.

```
POST _reindex
{
 "source": {
  "index": "source"
 },
 "dest": {
  "index": "dest",
  "pipeline": "some_ingest_pipeline"
 }
}
```

7.2.2.5.1. Reindex from a remote cluster

You can call the reindex operation to reindex from a remote Elasticsearch cluster.

```
POST _reindex
{
 "source": {
  "remote": {
   "host": "http://otherhost:9200",
   "username": "user",
   "password": "pass"
  },
  "index": "source",
  "query": {
   "match": {
    "test": "data"
   }
  }
 },
 "dest": {
  "index": "dest"
 }
}
```

The `host` parameter must contain a mode, host, and port number. A sample value is `https://otherhost:9200`). The `username` and `password` parameters are optional. If they are present, `_reindex` will connect to the remote Elasticsearch node by using basic authentication. When you use basic authentication, make sure that HTTPS is used. Otherwise, the password is sent in plaintext.

You must specify the `reindex.remote.whitelist` property to add the remote host to the whitelist in the elasticsearch.yml file. You can set the value to a comma-delimited list that contains combinations of allowed remote hosts and port numbers, for example, `otherhost:9200, another:9200, 127.0.10.*:9200, localhost:*` . The whitelist uses only the host and port number because the mode is omitted. Sample code:

```
reindex.remote.whitelist: "otherhost:9200, another:9200, 127.0.10.*:9200, localhost:*"
```

The whitelist must be configured on any nodes that are used to coordinate the reindex operation.

This feature applies to remote clusters deployed for Elasticsearch of any versions. This allows you to update Elasticsearch to the latest version by reindexing from a cluster of the source version.

To send queries to an early version of Elasticsearch, you can send the `query` parameter to the remote host without verification or modification.

> **Note** Reindexing from a remote cluster does not support manual slicing or automatic slicing.

Reindexing from a remote server uses an on-heap buffer with the default maximum size of 100 MB. If the remote index includes very large documents, you must specify a smaller batch size. In the following example, the batch size is set to `10` , which is very small.

```
POST _reindex
{
 "source": {
  "remote": {
   "host": "http://otherhost:9200"
  },
  "index": "source",
  "size": 10,
  "query": {
   "match": {
    "test": "data"
   }
  }
 },
 "dest": {
  "index": "dest"
 }
}
```

You can use the `socket_timeout` field to specify the socket read timeout period during remote connections. You can also use the `connect_timeout` field to specify the connection timeout period. The default values of the two fields are 30 seconds. In the following example, the socket_timeout field is set to 1m, and the connect_timeout field is set to 10s.

```
POST _reindex
{
 "source": {
  "remote": {
   "host": "http://otherhost:9200",
   "socket_timeout": "1m",
   "connect_timeout": "10s"
  },
  "index": "source",
  "query": {
   "match": {
    "test": "data"
   }
  }
 },
 "dest": {
  "index": "dest"
 }
}
```

7.2.2.5.2. URL parameters

In addition to standard parameters, such as `pretty`, the reindex operation supports the following parameters: `refresh`, `wait_for_completion`, `wait_for_active_shards`, `timeout`, `scroll`, and `requests_per_second`.

If the `refresh` parameter is specified in a request, Elasticsearch refreshes all indexes written by the request. This is different from the `refresh` parameter in an index operation. If the refresh parameter is specified in an index operation, Elasticsearch refreshes only shards that receive new data.

If a request contains `wait_for_completion=false`, Elasticsearch performs some prechecks and starts the request. Then, Elasticsearch returns a `task` that can be used together with the task operation to cancel the task or obtain the status of the task. Elasticsearch also creates a record of this task as a document in `.tasks/task/${taskId}`. You can retain or delete this document as needed. After the task is complete, delete the document so that Elasticsearch can reclaim the disk space occupied by the document.

The `wait_for_active_shards` parameter controls the number of replicas that must be active before Elasticsearch proceeds with a reindexing request. For more information, see Active shards. The `timeout` parameter specifies the period of time that a write request must wait for unavailable shards to become available. Both of the preceding parameters work the same way as they work in the bulk operation. Because `_reindex` uses scroll searches, you can specify the `scroll` parameter to control the period of time that is required to keep search contexts alive, such as `?scroll=10m`. The default value is 5m.

You can set the `requests_per_second` parameter to any positive decimal number, such as `1.4`, `6`, and `1000`, to throttle the rate at which `_reindex` issues batches of index operations. Each batch is padded with a wait time to throttle requests. You can set `requests_per_second` to `-1` to disable throttling.

Throttling is performed by waiting between batches so that `_reindex` can assign the internal `scroll` request a timeout period that takes the padding time into account. The padding time is the difference between the batch size divided by `requests_per_second` and the time that is required for a write operation. The default batch size is `1,000`. Therefore, if `requests_per_second` is set to `500`:

```
target_time = 1000 / 500 per second = 2 seconds
wait_time = target_time - write_time = 2 seconds - .5 seconds = 1.5 seconds
```

A batch is issued as a single `_bulk` request. If the size of a batch is large, Elasticsearch creates many requests and waits for a specific period of time before proceeding with the next batch. This is considered a burst operation instead of a smooth operation. The default value of requests_per_second is `-1`.

7.2.2.5.3. Response body

The following sample code provides a response in JSON format:

```
{
 "took": 639,
 "timed_out": false,
 "total": 5,
 "updated": 0,
 "created": 5,
 "deleted": 0,
 "batches": 1,
 "noops": 0,
 "version_conflicts": 2,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": 1,
 "throttled_until_millis": 0,
 "failures": [ ]
}
```

- `took`

  The total number of milliseconds that the operation took.
- `timed_out`

  If a request that is executed during reindexing times out, this field is set to `true`.
- `total`

  The number of documents that are processed.
- `updated`

  The number of documents that are updated.
- `created`

  The number of documents that are created.
- `deleted`

  The number of documents that are deleted.
- `batches`

  The number of scroll responses that are pulled back by the reindex operation.
- `noops`

The number of documents that are ignored because the script used for the reindex operation returns a `noop` value for `ctx.op`.

- `version_conflicts`

  The number of version conflicts incurred by the reindex operation.

- `retries`

  The number of retries attempted by the reindex operation. `bulk` indicates the number of bulk operations retried and `search` indicates the number of search operations retried.

- `throttled_millis`

  The number of milliseconds the request remains in the sleep state to conform to `requests_per_second`.

- `requests_per_second`

  The number of requests per second that are successfully executed during reindexing.

- `throttled_until_millis`

  The value of this field is always 0 in the response of the `_reindex` operation. This field is valid only when the task operation is called. In this case, this field indicates when (in milliseconds from epoch) the next throttling request will be executed to conform to `requests_per_second`.

- `failures`

  If unrecoverable errors occur during reindexing, an array of failures is returned. If this field is not empty, the request is stopped because of these failures. The reindex operation is implemented by using batches. Failures can cause the operation to stop, and all failures in the current batch are collected into the array. You can use the `conflicts` option to prevent the reindex operation from stopping because of version conflicts.

7.2.2.5.4. Task operation

You can call the task operation to obtain the status of all running reindex requests.

```
GET _tasks?detailed=true&actions=*reindex
```

Sample response:

```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
   "tasks" : {
    "r1A2WoRbTwKZ516z6NEs5A:36619" : {
     "node" : "r1A2WoRbTwKZ516z6NEs5A",
     "id" : 36619,
     "type" : "transport",
     "action" : "indices:data/write/reindex",
     "status" : {
      "total" : 6154,
      "updated" : 3500,
      "created" : 0,
      "deleted" : 0,
      "batches" : 4,
      "version_conflicts" : 0,
      "noops" : 0,
      "retries": {
       "bulk": 0,
       "search": 0
      },
      "throttled_millis": 0
     },
     "description" : ""
    }
   }
  }
 }
}
```

The `status` object indicates the actual status. It is the same as a response in JSON format, except the `total` field. The `total` field indicates the total number of operations performed by `_reindex`. You can add the `updated`, `created`, and `deleted` fields to estimate the progress. When the sum of the values for the updated, created, and deleted fields are equal to the value of the `total` field, the request is terminated.

You can specify the ID of a task to search for it. Example:

```
GET /_tasks/taskId:1
```

The task operation has the following advantages: It integrates with `wait_for_completion=false` and returns the status of completed tasks in a transparent manner. If tasks are complete and `wait_for_completion=false` is specified, the operation returns a `results` or `error` field. Note that `wait_for_completion=false` creates a document in `.tasks/task/${taskId}`. You can delete this document as needed.

7.2.2.5.5. Cancel tasks

You can call the task operation to cancel a reindexing task.

```
POST _tasks/task_id:1/_cancel
```

You can also call the task operation to obtain `task_id`.

The cancellation may require a few seconds. The task operation continues to return tasks until it is awakened to cancel itself.

7.2.2.5.6. Rethrottling

You can call the `_rethrottle` operation to change the value of `requests_per_second` for a running reindexing task.

```
POST _reindex/task_id:1/_rethrottle?requests_per_second=-1
```

You can also call the task operation to obtain `task_id`.

`requests_per_second` can be set to `-1`, which is the same as the setting for the reindex operation, to disable throttling. Alternatively, it can be set to any decimal number such as `1.7` or `12` to throttle the number of requests per second. Rethrottling that speeds up a query takes effect immediately. However, rethrottling that slows down a query takes effect after the current batch is processed. This prevents scroll timeouts.

7.2.2.5.7. Reindex to change a field name

You can call the `_reindex` operation to create a copy for an index and specify fields whose names you want to change. For example, you run the following command to create a document in an index:

```
POST test/_doc/1?refresh
{
 "text": "words words",
 "flag": "foo"
}
```

After the document is created, you want to change the field name `flag` to `tag`. You can call the `_reindex` operation to create another index.

```
POST _reindex
{
 "source": {
  "index": "test"
 },
 "dest": {
  "index": "test2"
 },
 "script": {
  "source": "ctx._source.tag = ctx._source.remove(\"flag\")"
 }
}
```

Run the following command to obtain the new document:

```
GET test2/_doc/1
```

The following result is returned:

```
{
 "found": true,
 "_id": "1",
 "_index": "test2",
 "_type": "_doc",
 "_version": 1,
 "_source": {
  "text": "words words",
  "tag": "foo"
 }
}
```

7.2.2.5.8. Slicing

The reindex operation allows you to perform reindexing in parallel with sliced scroll. During reindexing, you can split a request into slices. This improves the efficiency of the request.

Manual slicing

You can specify slice IDs and the total number of slices for each reindex request to manually split the request.

```
POST _reindex
{
 "source": {
  "index": "twitter",
  "slice": {
   "id": 0,
   "max": 2
  }
 },
 "dest": {
  "index": "new_twitter"
 }
}
POST _reindex
{
 "source": {
  "index": "twitter",
  "slice": {
   "id": 1,
   "max": 2
  }
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

You can run the following command to check whether the manual slicing succeeded:

```
GET _refresh
        POST new_twitter/_search?size=0&filter_path=hits.total
```

A result similar to the following code is returned. The code contains the `total` field with a reasonable value.

```
{
 "hits": {
  "total": 120
 }
}
```

Automatic slicing

You can also perform sliced scroll to split `_uid` so that the parallel `_reindex` operation can be automatically performed. Set the `slices` parameter to specify the number of slices to use.

```
POST _reindex?slices=5&refresh
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

You can run the following command to check whether the automatic slicing succeeded:

```
POST new_twitter/_search?size=0&filter_path=hits.total
```

A result similar to the following code is returned. The code contains the `total` field with a reasonable value.

```
{
 "hits": {
  "total": 120
 }
}
```

If the `slices` parameter is set to `auto`, Elasticsearch determines the number of slices to use. In this case, each shard uses one slice until the number of slices reaches the upper limit. If multiple source indexes exist, Elasticsearch sets the number of slices based on the index that has the least number of shards.

If you specify the `slices` parameter when you call the `_reindex` operation, manual slicing becomes automated. This generates subrequests that have different characteristics.

- You can call the task operation to view the subrequests. These subrequests are subtasks of request tasks that contain `slices`.
- When you query the status of request tasks that contain `slices`, only the status of the tasks that have completed slicing can be obtained.
- Subrequests can be separately processed for operations such as cancellation and rethrottling.
- If you use `slices` to rethrottle a request, only unfinished subrequests are proportionally rethrottled.
- If you use `slices` to cancel a request, all its subrequests are canceled.
- When `slices` is used, documents are not evenly allocated to subrequests. All documents are processed, but some slices may be greater than other slices. Greater slices may have a more even distribution.
- In requests that contain `slices`, the `requests_per_second` and `size` parameters are proportionally allocated to subrequests. Because the distribution is uneven, a request that contains `slices` and a specified `size` may not return a reindexed document that has the specified `size` value.
- The source index snapshots obtained by each subrequest are slightly different, but these snapshots are created at almost the same time.

Set the number of slices

For automatic slicing in which `slices` is set to `auto`, Elasticsearch sets a reasonable number of slices for most indexes. If you want to perform manual slicing or adjust automatic slicing, use the following guidelines:

If the value of `slices` is equal to the number of shards in an index, you can obtain the optimal query performance. If an index has a large number of shards, such as 500, set `slices` to a small number. Too many slices affect performance. Setting `slices` to a value greater than the number of shards does not improve efficiency but increases overheads.

Reindexing performance scales linearly across available resources based on the number of slices.

The documents that are being reindexed and cluster resources determine whether query or reindexing performance is dominant during the runtime.

7.2.2.5.9. Reindex daily indexes

You can use `_reindex` in combination with painless to reindex daily indexes and apply a new template to existing documents.

For example, your index consists of the following documents:

```
PUT metricbeat-2016.05.30/_doc/1?refresh
{"system.cpu.idle.pct": 0.908}
PUT metricbeat-2016.05.31/_doc/1?refresh
{"system.cpu.idle.pct": 0.105}
```

The new template of the `metricbeat-*` index is loaded to Elasticsearch, but this template applies only to newly created indexes. The painless operation can be used to reindex existing documents and apply new templates.

You can use the following script to extract the date from an index name and create an index with `-1` appended. All data from `metricbeat-2016.05.31` are reindexed into `metricbeat-2016.05.31-1`.

```
POST _reindex
{
 "source": {
  "index": "metricbeat-*"
 },
 "dest": {
  "index": "metricbeat"
 },
 "script": {
  "lang": "painless",
  "source": "ctx._index = 'metricbeat-' + (ctx._index.substring('metricbeat-'.length(), ctx._index.length())) + '-1'"
 }
}
```

After the script is executed, you can find all documents from the metricbeat index in the `*-1` index.

```
GET metricbeat-2016.05.30-1/_doc/1
GET metricbeat-2016.05.31-1/_doc/1
```

You can use the preceding script in combination with the method described in Perform reindexing to change the name of a field to add existing data to the new index and rename fields as needed.

7.2.2.5.10. Extract a random subset of an index

`_reindex` can be used to extract a random subset of an index for testing.

```
POST _reindex
{
 "size": 10,
 "source": {
  "index": "twitter",
  "query": {
   "function_score" : {
    "query" : { "match_all": {} },
    "random_score" : {}
   }
  },
  "sort": "_score"
 },
 "dest": {
  "index": "random_twitter"
 }
}
```

`sort` : The default value is `_doc` . If you want to sort documents by `_score` , you must change the default value to `_score` . Otherwise, `random_score` is invalid.

8.Elasticsearch (on k8s)

8.1. List of operations by function

Elasticsearch provides a RESTful API, which allows you to use HTTP API operations to manage Elasticsearch clusters. You can perform operations such as addition, deletion, modification, search, and alias configuration.

For more information, see Elasticsearch RESTful API.

Elasticsearch Reference [7.2]

Single-document operations

- Index
- Get
- Delete
- Update

Multi-document operations

- Multi get
- Bulk
- Delete by query
- Update by query
- Reindex

Use a Java REST client to access clusters

Only access over HTTP or TCP is supported. We recommend that you use a Java REST client provided by open source Elasticsearch.

Use the Java API to access clusters

Elasticsearch provides a built-in client for Java users to access clusters. For more information about the Java API, see Java API.

Transport Client

Transport Client allows you to send requests to a remote cluster. It is not a part of a cluster. It only forwards requests to nodes in a cluster.

Transport Client uses the Elasticsearch transport protocol to communicate with clusters over port `9300`.

Nodes in a cluster also use port `9300` to communicate with each other. Before you group nodes into a cluster, you must enable this port.

> **Note**
>
> Your nodes and Java clients must use the same Elasticsearch version. This way, the nodes and Java clients can identify each other.

RESTful API over HTTP

All other programming languages can use a RESTful API to communicate with Elasticsearch over port `9200`. You can use your preferred web client or the cURL tool to communicate with Elasticsearch.

> **Note**
>
> Open source Elasticsearch provides clients for several languages, including Groovy, JavaScript, .NET, PHP, Perl, Python, and Ruby.
>
> The open source community provides a number of clients and plug-ins. You can query the clients and plug-ins from open source Elasticsearch documentation.

Structure of a curl request over HTTP

```
curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

- *VERB*: the HTTP method. Valid values: GET, POST, PUT, HEAD, and DELETE.
- *PROTOCOL*: the protocol. Valid values: http and https. Use https only when HTTPS is enabled for Elasticsearch.
- *HOST*: the hostname of a node in your Elasticsearch cluster. If you use a local node, set this parameter to localhost.
- *PORT*: the port over which the Elasticsearch HTTP service runs. Default value: 9200.
- *PATH*: the endpoint of the API, such as `_count`. If _count is specified, the system returns the number of documents in a cluster. PATH can contain multiple segments, such as `_cluster/stats` and `_nodes/stats/jvm`.
- *QUERY_STRING*: the optional request parameters. For example, the ?pretty parameter makes the JSON-formatted data returned by the request easier to understand.
- *BODY*: the JSON-formatted request body. This field is required only when the request requires a body.

Examples

Run the following command to obtain the number of documents in an Elasticsearch cluster:

```
curl -XGET 'http://localhost:9200/_count?pretty' -d '
        {
        "query": {
        "match_all": {}
        }
        }'
```

Response:

```
{
        "count" : 0,
        "_shards" : {
        "total" : 5,
        "successful" : 5,
        "failed" : 0
        }
        }
```

Use the curl -i command to display the HTTP header:

```
curl -i -XGET 'localhost:9200/'
```

Complete sample request:

```
curl -XGET 'localhost:9200/_count?pretty' -d '
        {
        "query": {
        "match_all": {}
        }
        }'
```

Shortened sample request:

```
GET /_count
        {
        "query": {
        "match_all": {}
        }
        }
```

8.2. RESTful API

8.2.1. Single-document operations

8.2.1.1. Index

The index operation adds or updates a JSON document in a specific index to make the document searchable.

In the following example, the JSON document is inserted into the twitter index. The document type is _doc, and the document ID is 1.

```
PUT twitter/_doc/1
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

Response:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "result" : "created"
}
```

The `_shards` header provides information about the replication process of the index operation.

- `total` : indicates the number of shards and replicas on which the index operation is executed.
- `successful` : indicates the number of shards and replicas on which the index operation succeeds.
- `failed` : an array that contains replication errors when the index operation fails on replicas.

If the value of `successful` is greater than or equal to 1, the index operation succeeds.

> **Note** If the index operation succeeds, replicas may not all be started. Only shards must be started by default. This behavior can be changed. In this case, the value of `total` is equal to the total number of shards that is specified based on `number_of_replicas` , and the value of `successful` is equal to the total number of started shards and replicas. If no failures occur, the value of `failed` is 0.

8.2.1.1.1. Automatic index creation

The index operation automatically creates a specific index if the index does not exist. For more information about how to manually create an index, see Create index API. The index operation also automatically creates a dynamic mapping for a specific type if the mapping does not exist. For more information about how to manually create a type mapping, see Put mapping API.

Mappings are dynamic and do not require schemas. New fields and objects are automatically added to the mapping definition of a specific type. For more information about mapping definitions, see Mapping.

You can set `action.auto_create_index` to `false` in the configuration files of all nodes to disable automatic index creation. You can set `index.mapper.dynamic` to `false` to disable automatic mapping creation for each index.

Automatic index creation can contain a schema-based whitelist or blacklist. For example, you can set `action.auto_create_index` to `+aaa*,-bbb*,+ccc*,-*`. The plus sign (+) indicates allowed and the hyphen (-) indicates disallowed.

8.2.1.1.2. Versioning

Each indexed document has a version number. All associated `version` numbers are returned in the response to the index request. When you specify the `version` parameter, the index operation can determine whether to enable optimistic concurrency control. This controls the version of the document on which the index operation is intended to be executed. Versioning can be applied to scenarios such as performing a transactional read-then-update operation. You can specify `version` for the original document. This ensures that no changes will occur during the read process. If you want to update the document during the read process, we recommend that you set the `preference` parameter to `_primary`. Example:

```
PUT twitter/_doc/1?version=2
{
  "message" : "elasticsearch now has versioning support, double cool!"
}
```

> **Note** Versioning is implemented in real time and cannot be affected by near-real-time search operations. If no version is provided, the index operation is executed without version checks.

By default, internal versioning starts from 1 and increases with each update, including deletion. The version number can be supplemented with an external value in some cases, for example, the version number is maintained in a database. To enable this feature, set `version_type` to `external`. The provided value must be a long number that is greater than or equal to 0 and less than 9.2e + 18. When an external version type is used, the system does not check for a matching version number. Instead, it checks whether the version number that is transmitted to the index request is greater than the version number of the currently stored document. If the check result is true, the system indexes the document and uses the new version number for the document. If the provided value is less than or equal to the version number of the stored document, a version conflict occurs and the index operation fails.

> **Warning** External versioning considers 0 as a valid version number. This allows the version number to be synchronized with an external versioning system where version numbers start from 0 rather than 1. The disadvantage is that documents whose version numbers are 0 cannot be updated by calling the update by query operation or be deleted by calling the delete by query operation.

However, if version numbers from a source database are used, strict ordering is not required for asynchronized index operations that are executed as a result of changes to the source database. If external versioning is used, updating indexes by using data from a database is simplified. This is because only the latest version is used if the index operation fails.

Version types

In addition to the `internal` and `external` version types, Elasticsearch supports other version types for specific scenarios. The following content describes the supported version types and related semantics:

- `internal` : A document is indexed only when the provided version is the same as the version of the stored document.
- `external` or `external_gt` : A document is indexed only when the provided version is later than the version of the stored document or the document does not exist. The provided version is used as the new version and stored with the new document. The provided version must be a non-negative long number.
- `external_gte` : A document is indexed only when the provided version is later than or the same as the version of the stored document. If the document does not exist, the operation still succeeds. The provided version is used as the new version and stored with the new document. The provided version must be a non-negative long number.

> **Note** The `external_gte` version type is used for specific use cases. Proceed with caution. Improper use may cause data loss. The `force` version type is deprecated because it may cause inconsistency between shards and replicas.

8.2.1.1.3. Operation type

The index operation supports the `op_type` parameter to force a `create` operation, which allows the "put-if-absent" behavior. If the operation type is set to `create` and a document that has the same ID already exists in a specific index, the index operation fails.

Example:

```
PUT twitter/_doc/1?op_type=create
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

To set the operation type to `create`, you can also use the following URI:

```
PUT twitter/_doc/1/_create
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

8.2.1.1.4. Automatic ID generation

The index operation can be executed with no ID specified. In this case, the index operation automatically generates an ID. In addition, the `op_type` parameter is automatically set to `create`. Example (POST used instead of PUT):

```
POST twitter/_doc/
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

Response:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "W0tpsmIBdwcYyG50zbta",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "result": "created"
}
```

8.2.1.1.5. Routing

Shard placement or `routing` is controlled by using the hash of a document ID by default. To implement more precise control, you can use the `routing` parameter to specify the value that is fed into the hash function used by the router in each operation. Example:

```
POST twitter/_doc?routing=kimchy
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

In the preceding example, the _doc document is routed to a shard named kimchy based on the specified `routing` parameter.

When you create explicit mappings, you can use the `_routing` field to direct the index operation to extract the routing value from the document. This process requires a short period to parse the document and transmit the data. If the `_routing` mapping is defined and set to `required`, but the routing value is not specified or extracted, the index operation fails.

8.2.1.1.6. Distribution

The index operation is directed to a shard based on its routing. For more information, see the "Routing" topic. Then, it is executed on the actual node that stores this shard. After the shard completes the operation, updates are distributed to appropriate replicas if required.

8.2.1.1.7. Wait for active shards

To improve the flexibility of data write to the system, you can configure the index operation to proceed after the number of replicas for active shards reaches the specified threshold. If the replicas of the required number are unavailable, the write operation must wait and retry until the replicas are started or a timeout occurs. The default value of `wait_for_active_shards` is 1. This indicates that the write operation can proceed only when shards are active. You can specify `index.write.wait_for_active_shards` to dynamically overwrite the default value in index settings. To change this behavior of a specific operation, you can specify the `wait_for_active_shards` parameter in the relevant request.

This parameter can be set to `all` or any positive integer. The maximum value of this parameter is the total number of replicas per shard in an index. The total number of replicas per shard is `number_of_replicas` plus one. If a negative number or a number greater than the total number of replicas per shard is specified, an error will occur.

For example, you have a cluster that contains three nodes: `A`, `B`, and `C`. You create an index and set `number of replicas` to 3. In this case, the total number of configured replicas per shard is four, which is one more than the number of nodes. If you call the index operation, the operation proceeds only when each shard is available by default. If `A` hosts shards, the index operation still proceeds even if `B` and `C` fail. If you set `wait_for_active_shards` to `3` in the request and all the three nodes are running as expected, the index operation can proceed only when three replicas of each shard are active. This is because three nodes are active and each node must store one replica. However, if you set `wait_for_active_shards` to `all` or `4`, the index operation will not proceed because not all four replicas of each shard are active in the index. The operation will time out unless a new node is added to the cluster to host the fourth replica of each shard.

Note that this setting only reduces the possibility that the write operation does not write the required number of replicas. It cannot avoid this scenario because checks for the number of replicas are performed only before the write operation starts. After the write operation starts, replication may fail on some replicas but succeeds on shards. The number of replicas on which replication succeeds or fails are displayed in the `_shards` part of the response for the write operation.

```
{
  "_shards":{
    "total":2,
    "failed":0,
    "successful":2
  }
}
```

8.2.1.1.8. Refresh

This parameter determines when the changes made by a request are searchable. For more information, see Refresh.

8.2.1.1.9. Noop updates

When you call the index operation to update a document, a new version of the document is always created even if you do not update the document. If you want to enable the detection of noop updates, you can call the `_update` operation and set `detect_noop` to true. Noop updates are unavailable for the index operation because the index operation does not obtain old source code and cannot compare it with new source code.

You can enable or disable noop updates as needed. When you perform this operation, you must take several factors into account. The factors include the frequency at which the data source sends updates that are actually noops and the number of queries per second on the shard that receives the updates.

8.2.1.1.10. Timeout

When you call the index operation, the shard that is assigned to execute the index operation may be unavailable. The possible reasons are that the shard is recovering from a gateway or being reallocated. By default, the index operation waits for a maximum of one minute on the shard. If the shard is still unavailable after one minute, the index operation fails and an error is returned. The `timeout` parameter can be used to specify how long the operation waits. In the following example, the timeout parameter is set to five minutes.

```
PUT twitter/_doc/1?timeout=5m
{
  "user" : "kimchy",
  "post_date" : "2009-11-15T14:12:12",
  "message" : "trying out Elasticsearch"
}
```

8.2.1.2. Get

This topic describes the get operation.

You can call the get operation to obtain a JSON document from an index based on the document ID. In the following example, a JSON document is obtained from the twitter index. The document type is _doc, and the document ID is 0.

```
GET twitter/_doc/0
```

Response:

```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "0",
  "_version" : 1,
  "found": true,
  "_source" : {
    "user" : "kimchy",
    "date" : "2009-11-15T14:12:12",
    "likes": 0,
    "message" : "trying out Elasticsearch"
  }
}
```

The response contains the `_index` , `_type` , `_id` , and `_version` information of the document that you want to obtain. If the document is found, the value of the `found` field is true, and the `_source` field contains the content of the document.

The get operation also allows you to use `HEAD` to check whether a document exists. Example:

```
HEAD twitter/_doc/0
```

8.2.1.2.1. Real-time get operation

The get operation is implemented in real time by default. It is not affected by the index refresh rate if data is searchable. If a document is updated but not refreshed, the get operation issues a refresh call to make the document searchable. This will also cause other documents that have been modified since the last refresh call to become searchable. To disable real-time get operations, you can set the `realtime` parameter to `false` .

8.2.1.2.2. Source filtering

By default, the get operation returns the content of the `_source` field unless the `stored_fields` parameter is specified or the `_source` field is disabled. You can run the following command to disable `_source` retrieval:

```
GET twitter/_doc/0?_source=false
```

The `_source` parameter is set to false.

If you need only one or two fields in `_source` , you can use the `_source_include` or `_source_exclude` parameter to include or exclude fields as needed. The two parameters are useful for partial retrieval of large documents because the parameters can reduce network overheads. Both parameters use a list of fields that are separated with commas (,) or use an expression that contains wildcards. Example:

```
GET twitter/_doc/0?_source_include=*.id&_source_exclude=entities
```

If you want to specify only the _source_include parameter, you can use a shorter notation. Example:

```
GET twitter/_doc/0?_source=*.id,retweeted
```

8.2.1.2.3. Stored fields

You can use the `stored_fields` parameter to specify a set of stored fields that you want to obtain. Requested fields that are not stored are ignored. For example, you can consider the following mapping:

```
PUT twitter
{
  "mappings": {
    "_doc": {
      "properties": {
        "counter": {
          "type": "integer",
          "store": false
        },
        "tags": {
          "type": "keyword",
          "store": true
        }
      }
    }
  }
}
```

Run the following command to add a document:

```
PUT twitter/_doc/1
{
  "counter" : 1,
  "tags" : ["red"]
}
```

Run the following command to retrieve the added document:

```
GET twitter/_doc/1?stored_fields=tags,counter
```

Response:

```
{
  "_index": "twitter",
  "_type": "_doc",
  "_id": "1",
  "_version": 1,
  "found": true,
  "fields": {
    "tags": [
      "red"
    ]
  }
}
```

Field values that are obtained from the document are always returned as an array. The `counter` field is not stored. Therefore, when the get operation attempts to obtain `stored_fields`, it ignores the counter field.

You can also retrieve metadata fields such as `_routing`. Example:

```
PUT twitter/_doc/2?routing=user1
{
  "counter" : 1,
  "tags" : ["white"]
  }
```

```
GET twitter/_doc/2?routing=user1&stored_fields=tags,counter
```

Response:

```
{
  "_index": "twitter",
  "_type": "_doc",
  "_id": "2",
  "_version": 1,
  "_routing": "user1",
  "found": true,
  "fields": {
    "tags": [
      "white"
    ]
  }
}
```

Only leaf fields can be returned by using the `stored_field` option. Object fields cannot be returned and such requests will fail.

8.2.1.2.4. Obtain only the _source field

You can use the endpoint `/{index}/{type}/{id}/_source` to obtain only the `_source` field of a document. Example:

```
GET twitter/_doc/1/_source
```

You can also use source filtering parameters to specify the parts of `_source` to return. Example:

```
GET twitter/_doc/1/_source?_source_include=*.id&_source_exclude=entities'
```

You can use HEAD with the endpoint for _source to check whether the _source field exists. Example:

```
HEAD twitter/_doc/1/_source
```

If the _source field is disabled in the mappings setting, the document does not contain this field.

8.2.1.2.5. Routing

If the routing feature is used during indexing, you must specify the routing value to obtain a document. Example:

```
GET twitter/_doc/2?routing=user1
```

The preceding request obtains a document whose ID is `2` from the twitter index. However, the request will be routed based on the user indicated by routing. Note that the document cannot be obtained if you specify an invalid value for routing.

8.2.1.2.6. Preference

The `preference` parameter specifies the shard or replica on which the get operation is executed. By default, the operation is randomly executed among all shards and replicas.

Valid values for the `preference` parameter:

- `_primary` : The operation will be executed only on shards.
- `_local` : The operation will be executed on a locally allocated shard if required.

Custom (string) value

A custom value ensures that the same shards are used for this custom value. If different shards in different refresh states are available, only shards with the same custom value are used. The custom value can be a web session ID or a username.

8.2.1.2.7. Refresh

You can set the `refresh` parameter to `true` to refresh the related shard before the get operation is executed and make the shard searchable. Before you set it to `true` , verify that the setting does not cause system overload but slows down indexing.

8.2.1.2.8. Distribution

The get operation is hashed into a specific shard ID. It is then redirected to one of the replicas within the shard ID and returns a result. The replicas are the shard and its replicas within that shard ID. The more replicas, the better scaling performance of the get operation.

8.2.1.2.9. Versioning support

You can use the `version` parameter to retrieve a document only when the version of the document is the same as the specified version. This applies to all version types, except `FORCE` . Documents of this version type are always retrieved. The `FORCE` version type is already deprecated.

Elasticsearch has marked existing documents as deleted and added new documents. The documents that are marked as deleted still exist, but you cannot access them. If you continue to index data, Elasticsearch removes the documents in the background.

8.2.1.3. Delete

You can call the delete operation to delete a JSON document from a specific index based on the document ID. In the following example, a JSON document is deleted from the `twitter` index. The document type is `_doc` , and the document ID is `1` .

```
DELETE /twitter/_doc/1
```

Response:

```
{
  "_shards" : {
    "total" : 2,
    "failed" : 0,
    "successful" : 2
  },
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_primary_term": 1,
  "_seq_no": 5,
  "result": "deleted"
}
```

8.2.1.3.1. Versioning

Each indexed document is versioned. When you delete a document, you can specify the `version` parameter to ensure that the document that you want to delete is actually being deleted and it is not modified during the deletion. Each write operation (including the deletion operation) executed on a document causes its version to be incremented. The version number of a deleted document remains available for a short period after deletion to allow control of concurrent operations. The duration that is required for the version of a deleted document to remain available depends on the setting of `index.gc_deletes`. The default value is 60 seconds.

8.2.1.3.2. Routing

If the routing feature is used during indexing, you must specify the routing value to delete a document. Example:

```
DELETE /twitter/_doc/1?routing=kimchy
```

The preceding request deletes a document whose ID is `1` from the twitter index. However, the request will be routed based on the user indicated by routing. Note that the document cannot be deleted if you specify an invalid value for routing.

If the `_routing` mapping is set to `required` but no routing value is specified, the delete operation will return `RoutingMissingException` and reject the request.

8.2.1.3.3. Automatic index creation

When an external versioning variant is used, the delete operation automatically creates a specific index if the index does not exist. For more information about how to manually create an index, see Create index API. The delete operation also automatically creates a dynamic mapping for a specific type if the mapping does not exist. For more information about how to manually create a type mapping, see Put mapping API.

8.2.1.3.4. Distribution

The delete operation is hashed into a specific shard ID. It is then redirected to a shard within that ID group and replicated (if required) to the replica of the shard within that ID group.

8.2.1.3.5. Wait for active shards

When you call the delete operation, you can specify the `wait_for_active_shards` parameter. This ensures that the delete operation is processed after the number of replicas for active shards reaches the minimum number. For more information, see Index API.

8.2.1.3.6. Refresh

This parameter specifies when the changes made by a request are refreshed and become searchable. For more information, see *?refresh* .

8.2.1.3.7. Timeout

When you call the delete operation, the shard that is assigned to perform the delete operation may be unavailable. The possible reasons are that the shard is recovering from a storage medium or being reallocated. The delete operation waits for a maximum of one minute on the shard by default. If the shard is still unavailable after one minute, the delete operation fails and an error is returned. The `timeout` parameter can be used to specify how long the operation waits. In the following example, the timeout parameter is set to five minutes.

```
DELETE /twitter/_doc/1?timeout=5m
```

8.2.1.4. Update operation

You can call the update operation to update a document based on a provided script. The operation obtains the document (collocated with the shard) from the index, runs the script (with optional script language and parameters), and reindexes the result (this operation can be deleted or ignored). It uses versioning to ensure that no updates have happened during the Get and Reindex operations.

Note: The update operation reindexes the entire document. It removes some network roundtrips and reduces chances of version conflicts between the Get and Index operations. You must enable the `_source` field to ensure that this operation works properly.

The following code provides an example of indexing a simple document:

```
PUT test/_doc/1
{
  "counter" : 1,
  "tags" : ["red"]
}
```

8.2.1.4.1. Perform a scripted update

To increase the counter, run the following command with a script specified:

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "ctx._source.counter += params.count",
    "lang": "painless",
    "params" : {
      "count" : 4
    }
  }
}
```

To add a tag to the tag list, run the following command with a script specified. If the tag already exists, it is still added.

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "ctx._source.tags.add(params.tag)",
    "lang": "painless",
    "params" : {
      "tag" : "blue"
    }
  }
}
```

In addition to the `_source` field, you can use the `ctx` mapping to obtain the following variables: `_index`, `_type`, `_id`, `_version`, `_routing`, and `_now`. The _now variable indicates the current timestamp.

You can add a field to the document. Example:

```
POST test/_doc/1/_update
{
  "script" : "ctx._source.new_field = 'value_of_new_field'"
}
```

You can also remove a field from the document. Example:

```
POST test/_doc/1/_update
{
  "script" : "ctx._source.remove('new_field')"
}
```

In addition, you can change the operation that is being executed. If the `tags` field contains `green`, the request deletes the document. Otherwise, the request performs `noop` operations.

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "if (ctx._source.tags.contains(params.tag)) { ctx.op = 'delete' } else { ctx.op = 'none' }",
    "lang": "painless",
    "params" : {
      "tag" : "green"
    }
  }
}
```

8.2.1.4.2. Perform a partial update

The update operation allows you to transmit a part of one document, which will be merged into an existing document. The following operations are included in the merging: simple recursive merging, inner merging of objects, and replacement of core key-value pairs and arrays. To replace the entire existing document, you must use the `index` operation instead. In the following example, a field is added to an existing document:

```
POST test/_doc/1/_update
{
  "doc" : {
    "name" : "new_name"
  }
}
```

If you specify both `doc` and `script`, `doc` is ignored. The best practice is to place the field pairs from a part of the document in script.

8.2.1.4.3. Detect noop updates

If you specify `doc`, its value is merged with the existing `_source`. By default, if the update operation detects an update with no changes, it returns "result": "noop". Example:

```
POST test/_doc/1/_update
{
  "doc" : {
    "name" : "new_name"
  }
}
```

If the value of `name` is `new_name` before the request is sent, the request is ignored. In this case, `noop` is returned in the `result` element of the response.

```
{
  "_shards": {
    "total": 0,
    "successful": 0,
    "failed": 0
  },
  "_index": "test",
  "_type": "_doc",
  "_id": "1",
  "_version": 6,
  "result": "noop"
}
```

If you want to disable the detection of noop updates, you can specify "detect_noop": false. Example:

```
POST test/_doc/1/_update
{
  "doc" : {
    "name" : "new_name"
  },
  "detect_noop": false
}
```

8.2.1.4.4. Upsert

If a document does not exist, the content of the `upsert` element is inserted as a new document. If the document exists, the following `script` is run:

```
POST test/_doc/1/_update
{
  "script" : {
    "source": "ctx._source.counter += params.count",
    "lang": "painless",
    "params" : {
      "count" : 4
    }
  },
  "upsert" : {
    "counter" : 1
  }
}
```

scripted_upsert

If you want to run the script regardless of whether the document exists (the script instead of the `upsert` element initializes the document), set `scripted_upsert` to `true`.

```
POST sessions/session/dh3sgudg8gsrgl/_update
{
  "scripted_upsert":true,
  "script" : {
    "id": "my_web_session_summariser",
    "params" : {
      "pageViewEvent" : {
        "url":"foo.com/bar",
        "response":404,
        "time":"2014-01-01 12:32"
      }
    }
  },
  "upsert" : {}
}
```

doc_as_upsert

If you set `doc_as_upsert` to `true` instead of sending a part of `doc` and the `upsert` doc, the content of `doc` is used as the value of `upsert`. Example:

```
POST test/_doc/1/_update
{
  "doc": {
    "name": "new_name"
  },
  "doc_as_upsert": true
}
```

8.2.1.4.5. Parameters

The update operation supports the following parameters:

- `retry_on_conflict`

  Another process may have updated the same document in between the get and indexing phases of the update operation. In this case, the update operation fails due to a version conflict. The `retry_on_conflict` parameter specifies the number of retries before a version conflict is reported.

- `routing`

  This parameter is used to route an update request to the correct shard and sets routing for an upsert request when the document being updated does not exist. It cannot be used to update the routing of an earlier document.

- `timeout`

  The timeout period of waiting for a shard to become available.

- `wait_for_active_shards`

  The number of shard replicas that must be active before the update operation can proceed. For more information, see Index API.

- `refresh`

  This parameter specifies when the changes made by a request are refreshed and become searchable. For more information, see *?refresh*.

- `_source`

  This parameter specifies whether and how the updated source code is returned in the response. The updated source code is not returned by default. For more information, see Source filtering.

- `retry_on_conflict`

  The update operation uses the internal versioning of Elasticsearch to ensure that a document is unchanged during the update. You can specify the `version` parameter. This ensures that the document is updated only when its version is matched.

> **Note** The update operation supports only internal versioning.
>
> The update operation does not support external (version types `external` and `external_gte`) or forced (version type `force`) versioning because this will cause Elasticsearch version numbers to be inconsistent with the external system. You can use the `index` operation for external versioning.

8.2.2. Multi-document operations

8.2.2.1. Multi get

The multi get operation allows you to obtain multiple documents based on an index, a type, an ID, and routing. The type and routing are optional. The response of the operation contains `docs`, which is an array of documents. In this array, the documents are sorted based on the order in which the information of the documents is specified in the multi get request. If a specific get operation fails, an object that contains this error is included in the response. The structure of the response for a successfully obtained document is the same as that in the response of the get operation.

Example:

```
GET /_mget
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1"
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2"
    }
  ]
}
```

The `mget` endpoint can be used based on an index. In this case, you do not need to specify the index in the request body. Example:

```
GET /test/_mget
{
  "docs" : [
    {
      "_type" : "_doc",
      "_id" : "1"
    },
    {
      "_type" : "_doc",
      "_id" : "2"
    }
  ]
}
```

The endpoint can also be used based on a type. Example:

```
GET /test/_doc/_mget
{
  "docs" : [
    {
      "_id" : "1"
    },
    {
      "_id" : "2"
    }
  ]
}
```

In this case, you can use the `ids` element to simplify the request. Example:

```
GET /test/_doc/_mget
{
  "ids" : ["1", "2"]
}
```

8.2.2.1.1. Source filtering

If the `_source` field is stored, it is returned for each document by default. Same as the get operation, you can use the `_source` parameter to retrieve only parts of `_source` or retrieve no data. You can also use the URL parameters `_source`, `_source_include`, and `_source_exclude` to specify default values for retrieval. If you do not specify retrieval conditions for a specific document, these default values are used.

Example:

```
GET /_mget
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "_source" : false
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2",
      "_source" : ["field3", "field4"]
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "3",
      "_source" : {
        "include": ["user"],
        "exclude": ["user.location"]
      }
    }
  ]
}
```

8.2.2.1.2. Fields

You can specify a storage field for each document to be retrieved, similar to the stored_fields parameter of the get operation. Example:

```
GET /_mget
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "stored_fields" : ["field1", "field2"]
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2",
      "stored_fields" : ["field3", "field4"]
    }
  ]
}
```

You can also specify the `stored_fields` parameter in a query string as a default value that is to be applied to all documents.

```
GET /test/_doc/_mget?stored_fields=field1,field2
{
  "docs" : [
    {
      "_id" : "1"
    },
    {
      "_id" : "2",
      "stored_fields" : ["field3", "field4"]
    }
  ]
}
```

❶

Returns `field1` and `field2` .

❷

Returns `field3` and `field4` .

8.2.2.1.3. Routing

You can specify the routing value as a parameter. Example:

```
GET /_mget?routing=key1
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "routing" : "key2"
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2"
    }
  ]
}
```

In this example, the document indicated by `test/_doc/2` will be obtained from the shard that corresponds to the `key1` routing key. The document indicated by `test/_doc/1` will be obtained from the shard that corresponds to the `key2` routing key.

8.2.2.1.4. Security

For more information, see *URL-based access control*.

8.2.2.2. Bulk

You can call the bulk operation to index or delete multiple documents at a time. This operation significantly increases the indexing speed.

> **Note**
>
> The following clients that are officially supported provide helpers to assist in bulk request processing and reindexing of documents:
>
> - Perl: See Search::Elasticsearch::Client::5_0::Bulk and Search::Elasticsearch::Client::5_0::Scroll.
> - Python: See elasticsearch.helpers.

The endpoint of the RESTful API is `/_bulk`. It expects to return the following data in newline delimited JSON (NDJSON) format:

```
action_and_meta_data\n
optional_source\n
action_and_meta_data\n
optional_source\n
....
action_and_meta_data\n
optional_source\n
```

> **Note** The data in the last line must end with a newline character `\n`. Each newline character can be preceded by a carriage return `\r`. To send a request to the endpoint of the RESTful API, you must set the `Content-Type` header to `application/x-ndjson`.

You can perform the following operations: `indexing`, `creation`, `deletion`, and `update`. If you perform `indexing` or `creation` operations, make sure that a piece of source code is included in the next line. The two operations have the same semantics as the `op_type` parameter in the standard index API. If a document with the same index or type already exists, the creation operation fails and the indexing operation adds a document or replaces the existing document with a new one when necessary. The `deletion` operation does not require a piece of source code in the next line. It has the same semantics as the standard deletion API. The `update` operation requires that the doc, upsert, and script options be specified in the next line.

If you upload text files by using `curl` commands, you must use the `--data-binary` flag instead of the `-d` plaintext. This is because the latter does not retain newline characters. Example:

```
$ cat requests
{ "index" : { "_index" : "test", "_type" : "_doc", "_id" : "1" } }
{ "field1" : "value1" }
$ curl -s -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/_bulk --data-binary "@requests"; e
cho
{"took":7, "errors": false, "items":[{"index":{"_index":"test","_type":"_doc","_id":"1","_version":1,"result":"
created","forced_refresh":false}}]}
```

This format uses the combination of a backslash (\) and a lowercase letter n (`\n`) as delimiters. You must make sure that the JSON operations and source code are not prettified. The following code provides an example of a correct sequence of commands involved in a bulk operation:

```
POST _bulk
{ "index" : { "_index" : "test", "_type" : "_doc", "_id" : "1" } }
{ "field1" : "value1" }
{ "delete" : { "_index" : "test", "_type" : "_doc", "_id" : "2" } }
{ "create" : { "_index" : "test", "_type" : "_doc", "_id" : "3" } }
{ "field1" : "value3" }
{ "update" : {"_id" : "1", "_type" : "_doc", "_index" : "test"} }
{ "doc" : {"field2" : "value2"} }
```

Sample response:

```
{
  "took": 30,
  "errors": false,
  "items": [
    {
      "index": {
        "_index": "test",
        "_type": "_doc",
        "_id": "1",
        "_version": 1,
        "result": "created",
        "_shards": {
          "total": 2,
          "successful": 1,
          "failed": 0
        },
        "status": 201,
        "_seq_no" : 0,
        "_primary_term": 1
      }
    },
    {
      "delete": {
        "_index": "test",
        "_type": "_doc",
        "_id": "2",
        "_version": 1,
        "result": "not_found",
        "_shards": {
          "total": 2,
```

```
      "successful": 1,
      "failed": 0
    },
    "status": 404,
    "_seq_no" : 1,
    "_primary_term" : 2
  }
},
{
  "create": {
    "_index": "test",
    "_type": "_doc",
    "_id": "3",
    "_version": 1,
    "result": "created",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "status": 201,
    "_seq_no" : 2,
    "_primary_term" : 3
  }
},
{
  "update": {
    "_index": "test",
    "_type": "_doc",
    "_id": "1",
    "_version": 2,
    "result": "updated",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "status": 200,
    "_seq_no" : 3,
    "_primary_term" : 4
```

```
    }
   }
  ]
}
```

The endpoints are `/_bulk`, `/{index}/_bulk`, and `{index}/{type}/_bulk`. If /{index}/_bulk or {index}/{type}/_bulk is provided, it will be used for bulk operations that do not explicitly specify the index or index/type unless otherwise instructed.

Note: Some operations are redirected to other shards on other nodes. Only `action_meta_data` is parsed on the receiver node to complete the processing as soon as possible.

Client libraries using this protocol should try to perform similar operations on the client, and reduce buffering as much as possible.

The response to a bulk operation involves large amounts of data in JSON format because the response contains the result of each operation that has been executed. The failure of a single operation does not affect the remaining operations.

The number of operations required in a single API call cannot be explicitly specified. You should attempt to use different settings to find the optimal one for your particular workload.

To call the HTTP operation, make sure that the client does not send HTTP chunks because this slows the processing speed.

Versioning

Each bulk item can include the version value indicated by the `version` field. It automatically follows the behavior of the indexing or deletion operation based on the `_version` mapping. It also supports `version_type`. For more information, see Versioning.

Routing

Each bulk item can include the routing value indicated by the `routing` field. It automatically follows the behavior of the indexing or deletion operation based on the `_routing` mapping.

Wait for active shards

To call a bulk operation, you can specify the `wait_for_active_shards` parameter to ensure that the bulk request is processed immediately after the active shard replicas reach the minimum. For more details and usage examples, see Index operation.

Refresh

You can specify this parameter to control the time when the changes made by a request are visible to search. For more information, see Refresh.

Update

When you perform the `update` operation, `retry_on_conflict` can be used as a field in the operation (not in the extra payload line), to specify how many times an update should be retried in the case of a version conflict.

The `update` operation payload supports the following options: `doc` (partial documents), `upsert`, `doc_as_upsert`, `script`, `params` (for scripts), `lang` (for scripts), and `_source`. For more information about the options, see the update documentation. Example of an update operation:

```
POST _bulk
{"update":{"_id":"1","_type":"_doc","_index":"index1","retry_on_conflict":3}}
{"doc":{"field":"value"}}
{"update":{"_id":"0","_type":"_doc","_index":"index1","retry_on_conflict":3}}
{"script":{"source":"ctx._source.counter += params.param1","lang":"painless","params":{"param1":1}},"upsert":{"counter":1}}
{"update":{"_id":"2","_type":"_doc","_index":"index1","retry_on_conflict":3}}
{"doc":{"field":"value"},"doc_as_upsert":true}
{"update":{"_id":"3","_type":"_doc","_index":"index1","_source":true}}
{"doc":{"field":"value"}}
{"update":{"_id":"4","_type":"_doc","_index":"index1"}}
{"doc":{"field":"value"},"_source":true}
```

Security

See *URL-based access control*.

8.2.2.3. Delete by query

In most cases, the `_delete_by_query` operation is used to delete each document that matches a query. Example:

```
POST twitter/_delete_by_query
{
 "query":{
  "match":{
   "message":"some message"
  }
 }
}
```

`"query"`: The query must be passed as a value to the `query` key in the same way as the query for the search operation. You can also use the `q` parameter in the same way as that for the search operation.

Sample response:

```
{
 "took" : 147,
 "timed_out": false,
 "deleted": 119,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "total": 119,
 "failures" : [ ]
}
```

The `_delete_by_query` operation obtains a snapshot of an index when the operation starts and deletes the document that is obtained by using `internal` versioning. If the document changes when the operation obtains the snapshot and deletes the document, a version conflict occurs. If versions match, the operation deletes the document.

> **Note** `Internal` versioning considers the value 0 as an invalid version number. Therefore, documents with this version number cannot be deleted by using the `_delete_by_query` operation. If you use this operation to delete such a document, the operation fails.

When `_delete_by_query` is executed, multiple search requests are executed in sequence to search for all matching documents that you want to delete. Each time documents are obtained, a bulk request is executed to delete the documents. If a search or bulk request is rejected, `_delete_by_query` uses exponential backoff to retry the request based on the default policy. A maximum of 10 retries are allowed. If the number of retries reaches the upper limit, `_delete_by_query` stops and returns all failures in the `failures` element of the response. The deletions that are performed remain valid. This indicates that the process is not rolled back. It is only stopped. Although the first failure causes the process to stop, all failures that are returned by the failed bulk request are included in the `failures` element. A large number of failures may be returned.

If you want to collect version conflicts but do not want to cause the process to stop, you can specify `conflicts=proceed` in the URL or `"conflicts": "proceed"` in the request body.

After you send a _delete_by_query request, tweets are deleted from the `twitter` index. Example:

```
POST twitter/_doc/_delete_by_query?conflicts=proceed
{
 "query": {
  "match_all": {}
 }
}
```

You can also delete documents of multiple indexes and multiple types at a time, in a similar way to the search operation. Example:

```
POST twitter,blog/_docs,post/_delete_by_query
{
 "query": {
  "match_all": {}
 }
}
```

If you specify `routing` in the request, it is replicated to a scroll query so that the process can be limited to the shards that match the routing value. Example:

```
POST twitter/_delete_by_query?routing=1
{
 "query": {
  "range" : {
   "age" : {
    "gte" : 10
   }
  }
 }
}
```

The default scroll batch size used by `_delete_by_query` is 1,000. You can use the `scroll_size` URL parameter to change the scroll batch size. Example:

```
POST twitter/_delete_by_query?scroll_size=5000
{
 "query":{
  "term":{
   "user":"kimchy"
  }
 }
}
```

8.2.2.3.1. URL parameters

In addition to standard parameters such as `pretty`, the _delete_by_query operation supports the following parameters: `refresh`, `wait_for_completion`, `wait_for_active_shards`, `timeout`, and `scroll`.

If the `refresh` parameter is specified, all the shards involved in the _delete_by_query request are refreshed immediately after the request is completed. This is different from the `refresh` parameter of the deletion operation. If the refresh parameter of the deletion operation is specified, only the shards that have received the deletion request are refreshed.

If the request contains `wait_for_completion=false`, Elasticsearch performs some prechecks, starts the request, and returns a task that can be used together with the task operation to cancel the task or query the status of the task. Elasticsearch also creates a record of this task as a document in `.tasks/task/${taskId}`. You can retain or delete this document as required. After you complete this operation, delete the document so that Elasticsearch can reclaim the disk space.

The `wait_for_active_shards` parameter controls how many replicas of a shard must be active before Elasticsearch proceeds with the request processing. For more information, see Index operation. The `timeout` parameter specifies the time period that a write request waits for unavailable shards to become available. The two operations work exactly the same way they work in the bulk operation. The `_delete_by_query` operation uses scrolled search. You can specify the `scroll` parameter to control how long it keeps the search context open, for example, `?scroll=10m`. The default value is 5m.

The `requests_per_second` parameter can be set to a positive decimal number, such as `1.4`, `6`, or `1000`. You can control the rate at which `_delete_by_query` issues batches of deletion operations by padding each batch with a wait time. You can set `requests_per_second` to `-1` to disable throttling.

A padding time between batches is used to throttle the rate so that the `_delete_by_query` operation can assign the internal scroll requests a timeout that takes the padding time into account. The padding time equals the difference between the batch size divided by `requests_per_second` and the time spent on writing. By default, the batch size is `1000`. If the `requests_per_second` parameter is set to `500`, the padding time is obtained based on the following calculation result:

```
target_time = 1000 / 500 per second = 2 seconds
wait_time = target_time - write_time = 2 seconds - .5 seconds = 1.5 seconds
```

The batch is issued as a single `_bulk` request. If the batch size is set to a large value, Elasticsearch has to create many requests and wait a period of time before proceeding with the next batch. This is "bursty" instead of "smooth". By default, requests_per_second is set to `-1`.

8.2.2.3.2. Response body

Sample response in JSON format:

```
{
 "took" : 147,
 "timed_out": false,
 "total": 119,
 "deleted": 119,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "failures" : [ ]
}
```

- `took`:

  The total number of milliseconds that the operation took.
- `timed_out`:

  If the requests executed during the _delete_by_query operation time out, this field is set to `true`.
- `total`:

  The number of documents that have been processed.
- `deleted`:

  The number of documents that have been deleted.
- `batches`:

The number of scroll responses revoked by the _delete_by_query operation.

- `version_conflicts` :

  The number of version conflicts caused by the _delete_by_query operation.

- `noops` :

  The value of this field is always zero in the response of the _delete_by_query operation. This field is reserved to ensure that the _delete_by_query, _update_by_query, and _reindex operations return responses in the same format.

- `retries` :

  The number of retries attempted by the _delete_by_query operation. `bulk` indicates the number of bulk operations retried and `search` indicates the number of search operations retried.

- `throttled_millis` :

  The number of milliseconds that the request was in the sleep state to conform to `requests_per_second` .

- `requests_per_second` :

  The number of requests per second that were executed during the _delete_by_query operation.

- `throttled_until_millis` :

  The value of this field is always zero in the response of the _delete_by_query operation. It is valid only when the delete by query operation is called. In this case, it indicates the time (in milliseconds since epoch) when the next throttling request will be executed to conform to `requests_per_second` .

- `failures` :

  An array of failures is returned if unrecoverable errors occur in this process. If this field is not empty, the request is stopped because of these failures. The _delete_by_query operation is implemented in batches. Any failures will cause the entire process to stop, but all failures in the current batch will be recorded in the array. You can use the `conflicts` option to prevent the _delete_by_query operation from stopping due to version conflicts.

8.2.2.3.3. Cancel a _delete_by_query task

You can call the task management operation to cancel a _delete_by_query tasks.

```
POST _tasks/task_id:1/_cancel
```

You can also call the task management operation to obtain `task_id` .

The cancellation may take several seconds. The task management operation continues to list the _delete_by_query task until this task checks that it has been canceled.

8.2.2.3.4. Obtain the status of running _delete_by_query requests

You can call the task management operation to obtain the status of running _delete_by_query requests.

```
GET _tasks? detailed=true&actions=*/delete/byquery
```

Sample response:

```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
   "tasks" : {
    "r1A2WoRbTwKZ516z6NEs5A:36619" : {
     "node" : "r1A2WoRbTwKZ516z6NEs5A",
     "id" : 36619,
     "type" : "transport",
     "action" : "indices:data/write/delete/byquery",
     "status" : {
      "total" : 6154,
      "updated" : 0,
      "created" : 0,
      "deleted" : 3500,
      "batches" : 36,
      "version_conflicts" : 0,
      "noops" : 0,
      "retries": 0,
      "throttled_millis": 0
     },
     "description" : ""
    }
   }
  }
 }
}
```

`status` : This object contains the actual status. It is similar to the response in JSON format. The only difference is that the important `total` field is added to status. The `total` field indicates the total number of operations that the reindexing operation expects to perform. You can add the `updated` , `created` , and `deleted` fields to estimate the progress. When the sum of the values for the updated, created, and deleted fields is equal to the value of the `total` field, the request is terminated.

You can search for a task based on its ID.

```
GET /_tasks/taskId:1
```

The task operation has the following advantages: It integrates with `wait_for_completion=false` and returns the status of completed tasks in a transparent manner. If tasks are complete and `wait_for_completion=false` is specified, the operation returns a `results` or `error` field. Note that `wait_for_completion=false` will create a document in `tasks/task/${taskId}` . You can delete this document as needed.

8.2.2.3.5. Rethrottling

You can call the `_rethrottle` operation to change the value of `requests_per_second` for any running _delete_by_query task.

```
POST _delete_by_query/task_id:1/_rethrottle? requests_per_second=-1
```

You can also call the _rethrottle operation to obtain `task_id` .

You can set `requests_per_second` to `-1` to disable throttling in the same way as in the `_delete_by_query` operation. You can also set requests_per_second to any decimal number, such as `1.7` or `12` to control throttling. Rethrottling that speeds up a _delete_by_query task takes effect immediately. However, rethrottling that slows down a _delete_by_query task takes effect only after the current batch is processed. This prevents scroll timeouts.

8.2.2.3.6. Slicing

The _delete_by_query operation allows you to perform deletion in parallel with sliced scroll. During deletion, you can split a request into slices. This improves the efficiency of request processing.

Manual slicing

You can specify slice IDs and the total number of slices for each _delete_by_query request to manually split the request.

```
POST twitter/_delete_by_query
{
 "slice": {
  "id": 0,
  "max": 2
 },
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
POST twitter/_delete_by_query
{
 "slice": {
  "id": 1,
  "max": 2
 },
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

You can use the following method to check whether manual slicing is successful:

```
GET _refresh
        POST twitter/_search? size=0&filter_path=hits.total
{
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

A similar response with a reasonable `total` value is displayed:

```
{
 "hits": {
  "total": 0
 }
}
```

Automatic slicing

You can also call the sliced scroll operation to split `_uid`, so that a parallel _delete_by_query operation is automatically performed. The `slices` parameter is used to specify the number of slices to use.

```
POST twitter/_delete_by_query? refresh&slices=5
{
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

You can use the following method to check whether automatic slicing is successful:

```
POST twitter/_search? size=0&filter_path=hits.total
{
 "query": {
  "range": {
   "likes": {
    "lt": 10
   }
  }
 }
}
```

A similar response with a reasonable `total` value is displayed:

```
{
 "hits": {
  "total": 0
 }
}
```

If the `slices` parameter is set to `auto`, Elasticsearch determines the number of slices to use. In this case, each shard uses one slice until the number of slices reaches the upper limit. If multiple source indexes exist, Elasticsearch sets the number of slices based on the index that has the minimum number of shards.

If `slices` is added to `_delete_by_query`, the manual slicing process becomes automated. This method generates subrequests with different characteristics.

- You can view these subrequests in the task operation. These subrequests are subtasks of the request tasks that contain `slices`.
- When you query the status of request tasks that contain `slices`, only the status of the tasks that have completed slicing can be obtained.
- These subrequests can be separately processed for throttling cancellation and rethrottling operations.
- If you use `slices` to rethrottle a request, only unfinished subrequests are rethrottled proportionally.
- If you use `slices` to cancel a request, all its subrequests are canceled.
- When `slices` is used, the document segments are not evenly allocated to each subrequest. All documents will be processed, but some slices may be greater than other slices. Greater slices may have a more even distribution.
- In requests that contain `slices`, the `requests_per_second` and `size` parameters are allocated to each subrequest proportionally. Based on the uneven distribution principle, a request that contains `slices` and a specified `size` may not return a deleted document that has the specified `size` value.
- The source index snapshots obtained by each subrequest are slightly different, but these snapshots

are taken at approximately the same time.

Configure the number of slices

For automatic slicing where `slices` is set to `auto`, Elasticsearch sets a reasonable number of slices for most indexes. Use the following guidelines if you want to perform manual slicing or adjust automatic slicing:

If the value of `slices` is equal to the number of shards in an index, you can obtain the optimal query performance. A value greater than the number of shards in an index, for example, 500, may not improve efficiency but increase overheads, resulting in performance degradation.

Deletion performance scales linearly across available resources with the number of slices.

Both the documents that are being reindexed and cluster resources determine whether query or deletion performance is dominant during the runtime.

8.2.2.4. Update By Query operation

The most basic usage of the `_update_by_query` operation is to perform updates on each document in an index. In this case, you do not need to change the source code. It is useful to obtain new properties or several other online mapping changes. Example:

```
POST twitter/_update_by_query? conflicts=proceed
```

The following or similar result is returned:

```
{
 "took" : 147,
 "timed_out": false,
 "updated": 120,
 "deleted": 0,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "total": 120,
 "failures" : [ ]
}
```

The `_update_by_query` operation obtains a snapshot of the index when the process of this operation starts and indexes the document that is found during `internal` versioning. Version conflicts may occur when a document changes. The document change occurs after the snapshot is obtained and before the index request is processed. If the versions match, the document is updated and the version number is incremented.

> **Note** The document with version number 0 cannot be updated by calling the `_update_by_query` operation. This is because `internal` versioning considers the value 0 as an invalid version number and the update requests from the document will fail.

All update and query failures cause the `_update_by_query` operation to stop and these failures are included in the `failures` element of the response. The _update_by_query operation that has been performed is still effective. This means that the process of this operation is not restarted but is stopped. Although the first failure causes the process to stop, all failures that are returned by the failed bulk request are included in the `failures` element. Therefore, a few failed entities may exist.

If you want to count only version conflicts without stopping the `_update_by_query` operation, set `conflicts=proceed` in the URL, or set `"conflicts": "proceed"` in the request body. In the preceding example of the _update_by_query operation, this setting is used because the `_update_by_query` operation only tries to obtain an online mapping change. Version conflicts may occur when a document changes. The document change occurs after the _update_by_query operation starts and before the change operations are performed. This is normal because the _update_by_query operation has obtained the online mapping change.

After the operation format is restored, tweets are updated from the `twitter` index.

```
POST twitter/_doc/_update_by_query? conflicts=proceed
```

You can also use Query DSL to query the `_update_by_query` operation. This way, all documents are updated from the `twitter` index for user `kimchy`.

```
POST twitter/_update_by_query? conflicts=proceed
{
 "query": {
  "term": {
   "user": "kimchy"
  }
 }
}
```

`"query"` must be passed as a value to the `query` key in the same way as the search operation. You can also use the `q` parameter in the same way as the search operation.

The preceding description demonstrates how to call the _update_by_query operation to update a document without the need to change the source code. This is a helpful operation for purposes such as obtaining new properties . You can also call the `_update_by_query` operation to update a document by using scripts. This will increment the `likes` field on all the tweets for user kimchy.

```
POST twitter/_update_by_query
{
 "script": {
  "source": "ctx._source.likes++",
  "lang": "painless"
 },
 "query": {
  "term": {
   "user": "kimchy"
  }
 }
}
```

Like the update operation, the _update_by_query operation allows you to specify `ctx.op` if you want to change the operation that is about to execute.

- `noop` : Set ctx.op to `noop` if your script determines that the document does not need to be changed. This way, the `_update_by_query` operation omits this documents from its updates. The `noop` counter reports "no operation" in the response body.
- `delete` : Set ctx.op to `delete` if your script determines that the document must be deleted. The `deleted` counter reports this deletion operation in the response body.

If you do not set `ctx.op` to noop or delete, an error is returned. The same rule applies to other fields in `ctx` .

Note that `conflicts=proceed` is not specified in this case. This is because we want to stop the process when a version conflict occurs.

The_update_by_query operation does not allow you to move the documents involved, but only allows you to modify their source code. No provisions are provided to remove the document from its original location.

You can also perform these operations on multiple indexes and multiple types at a time, in the same way as the search operation:

```
POST twitter,blog/_doc,post/_update_by_query
```

If you provide `routing` , the routing is copied to the scroll query, so that the process is limited to the shards that match the routing value.

```
POST twitter/_update_by_query? routing=1
```

By default, `_update_by_query` uses 1,000 scroll batches. You can use the `scroll_size` URL parameter to change the batch size.

```
POST twitter/_update_by_query? scroll_size=100
```

The `_update_by_query` operation also allows you to use an ingest node based on a specified `pipeline`, as shown in the following code:

```
PUT _ingest/pipeline/set-foo
{
 "description" : "sets foo",
 "processors" : [ {
   "set" : {
    "field": "foo",
    "value": "bar"
   }
 } ]
}
POST twitter/_update_by_query? pipeline=set-foo
```

8.2.2.4.1. URL parameters

In addition to the standard parameters such as `pretty`, the Update By Query operation supports the following parameters: `refresh`, `wait_for_completion`, `wait_for_active_shards`, `timeout`, and `scroll`.

If the `refresh` parameter is sent after a request is complete, all shards in the index that is being updated will be updated. This is different from the `refresh` parameter for the index operation. The latter one only indexes the shards that receive the new data.

If the request contains the `wait_for_completion=false` parameter, Elasticsearch runs some prechecks, initiates the request, and returns a task that can be used in combination with the task operation to cancel the task or query the status of the `task`. Elasticsearch can also create a record of this task as a document in `.tasks/task/${taskId}`. You can keep or delete the document as needed. After the task is complete, delete the document so that Elasticsearch can reclaim the disk space.

The `wait_for_active_shards` parameter controls how many replicas of a shard must be activated before Elasticsearch proceeds with the request. For more information, see Active shards. The `timeout` parameter controls how long each write request waits for unavailable shards to become available. Both parameters work exactly the same way they work inbulk operation. `_update_by_query` uses scroll search. You can set the `scroll` parameter to control how long it keeps the search context alive, for example, `? scroll=10m`. The default value is 5m.

The `requests_per_second` parameter can be set to a positive decimal number, for example, `1.4`, `6`, or `1000`. This parameter is used together with the wait time to throttle the rate at which `_update_by_query` issues batches of index operations. You can set `requests_per_second` to `-1` to disable throttling.

A padding time between batches is used to throttle the rate so that the `_update_by_query` operation can assign the internal scroll requests a timeout that takes the padding time into account. The padding time equals the difference between the batch size divided by `requests_per_second` and the time spent on writing. By default, the batch size is `1000`. If the `requests_per_second` parameter is set to `500`, the padding time is obtained based on the following calculation result:

```
target_time = 1000 / 500 per second = 2 seconds
wait_time = target_time - delete_time = 2 seconds - .5 seconds = 1.5 seconds
```

The batch is issued as a single `_bulk` request. Large batch sizes will cause Elasticsearch to create many requests and then wait for a while before starting the next batch. This is "bursty" instead of "smooth". By default, requests_per_second is set to `-1`.

8.2.2.4.2. Response body

Sample response in JSON format:

```
{
 "took" : 147,
 "timed_out": false,
 "total": 5,
 "updated": 5,
 "deleted": 0,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "failures" : [ ]
}
```

- `took`: the total number of milliseconds that the operation took.

- `timed_out` : If a request times out during the_update_by_query operation, this field is set to `true` .
- `Total` : the number of documents that have been processed.
- `updated` : the number of documents that have been updated.
- `deleted` : the number of documents that have been deleted.
- `batches` : the number of scroll responses revoked by the _update_by_query operation.
- `version_conflicts` : the number of version conflicts caused by the _update_by_query operation.
- `noops` : the number of documents that were omitted because the script used for the update by query returned a `noop` value for `ctx.op` .
- `retries` : the number of retries attempted by the update by query operation. `bulk` indicates the number of bulk operations retried. `search` indicates the number of search operations retried.
- `throttled_millis` : the number of milliseconds that the request is in sleep mode to conform to `reques ts_per_second` .
- `requests_per_second` : the number of requests per second that are effectively executed during the _update_by_query operation.
- `throttled_until_millis` : This field must be zero in a delete by query response. It is valid only when the task operation is called. In this situation, this field indicates that a throttling request will be executed again to conform to `requests_per_second` since epoch in milliseconds.
- `failures` : An array of failures is returned if unrecoverable errors occur in this process. If this field is not empty, the request is terminated because of these failures. The _update_by_query operation is implemented in batches. Any failures will cause the entire process to stop, but all failures in the current batch will be recorded in the array. You can use the `conflicts` option to prevent the _update_by_query operation from stopping due to version conflicts.

8.2.2.4.3. Obtain the status of running update-by-query requests

You can call the task operation to obtain the status of running update-by-query requests.

```
GET _tasks? detailed=true&actions=*byquery
```

Sample response:

```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
   "tasks" : {
    "r1A2WoRbTwKZ516z6NEs5A:36619" : {
     "node" : "r1A2WoRbTwKZ516z6NEs5A",
     "id" : 36619,
     "type" : "transport",
     "action" : "indices:data/write/update/byquery",
     "status" : {
      "total" : 6154,
      "updated" : 3500,
      "created" : 0,
      "deleted" : 0,
      "batches" : 4,
      "version_conflicts" : 0,
      "noops" : 0,
      "retries": {
       "bulk": 0,
       "search": 0
      }
      "throttled_millis": 0
     },
     "description" : ""
    }
   }
  }
 }
}
```

`"status"` : The object contains the actual status. It is similar to the response in JSON format. The only difference is that the important `total` field is added to status. The `total` field indicates the total number of operations that the _update_by_query operation expects to perform. You can add the `updated` , `created` , and `deleted` fields to estimate the progress. The request will finish when the sum is equal to the value of the `total` field.

You can search for a task based on its ID:

```
GET /_tasks/taskId:1
```

The advantage of this API is that it integrates with the `wait_for_completion=false` to return the status of completed tasks in a transparent manner. If the task is completed and `wait_for_completion=false` is set for it, this task will return the `results` or `error` field. Note that `wait_for_completion=false` will create a document in `.tasks/task/${taskId}` . You can delete this document as needed.

8.2.2.4.4. Cancel tasks

You can call the task management operation to cancel an update by query task.

```
POST _tasks/task_id:1/_cancel
```

You can also call the task management operation to obtain `task_id` .

The cancellation may take several seconds. The task management operation continues to list the update by query task until this task checks that it has been canceled.

8.2.2.4.5. Rethrottling

You can call the `_rethrottle` operation to change the value of `requests_per_second` on a running update by query task.

```
POST _update_by_query/task_id:1/_rethrottle? requests_per_second=-1
```

You can also call the task management operation to obtain `task_id` .

You can set `requests_per_second` to -1 to disable throttling in the same way as in the `_update_by_query` operation. Alternatively, you can set this field to a decimal number such as `1.7` or `12` to control throttling. Rethrottling that speeds up an update by query task takes effect immediately. However, rethrottling that slows down an update by query task takes effect only after the current batch is processed. This prevents scroll timeouts.

8.2.2.4.6. Slicing

You can call the _update_by_query operation and the sliced scroll operation at the same time. Simultaneous operations allow you to split requests into smaller parts. This improves the efficiency of request processing.

Manual slicing

You can specify slice IDs and the total number of slices for each _update_by_query request to manually split the request.

```
POST twitter/_update_by_query
{
 "slice": {
  "id": 0,
  "max": 2
 },
 "script": {
  "source": "ctx._source['extra'] = 'test'"
 }
}
POST twitter/_update_by_query
{
 "slice": {
  "id": 1,
  "max": 2
 },
 "script": {
  "source": "ctx._source['extra'] = 'test'"
 }
}
```

You can use the following method to check whether manual slicing is successful:

```
GET _refresh
        POST twitter/_search? size=0&q=extra:test&filter_path=hits.total
```

A similar response with a suitable `total` value appears:

```
{
 "hits": {
  "total": 120
 }
}
```

Automatic slicing

You can also call the sliced scroll operation to split `_uid`, so that a parallel _update_by_query

operation is automatically performed. `slices` is used to specify the number of slices you use.

```
POST twitter/_update_by_query? refresh&slices=5
{
 "script": {
  "source": "ctx._source['extra'] = 'test'"
 }
}
```

You can use the following method to check whether automatic slicing is successful:

```
POST twitter/_search? size=0&q=extra:test&filter_path=hits.total
```

A similar response with a suitable `total` value appears:

```
{
 "hits": {
  "total": 120
 }
}
```

If the `slices` parameter is set to `auto`, Elasticsearch will determine the number of slices to use. In this case, each shard uses one slice until the number of slices reaches the upper limit. If multiple source indexes exist, Elasticsearch will set the number of slices based on the index that has a minimum number of shards.

If `slices` is added to `_update_by_query`, the manual slicing process becomes automated. This method generates subrequests with different characteristics.

- You can view these subrequests in the task operation. These subrequests are subtasks of the request tasks that contain `slices`.
- When you query the status of request tasks that contain `slices`, only the status of the tasks that have completed slicing can be obtained.
- These subrequests can be separately processed for throttling cancellation and rethrottling operations.
- If you use `slices` to rethrottle a request, only unfinished subrequests are proportionally rethrottled.
- If you use `slices` to cancel a request, all its subrequests are canceled.
- When `slices` is used, the document segments are not evenly allocated to each subrequest. All documents will be processed. However, some slices may be greater than other slices. Greater slices may have a more even distribution.
- In requests that contain `slices`, the `requests_per_second` and `size` parameters are proportionally allocated to each subrequest. Based on the uneven distribution principle, a request that contains `slices` and a specified `size` may not return an updated document that has the specified `size` value.

- The source index snapshots obtained by each subrequest are slightly different, but these snapshots are taken at approximately the same time.

Choose the number of slices

For automatic slicing where `slices` is set to `auto`, Elasticsearch sets a suitable number of slices for most indexes. If you want to use manual slicing or other methods to adjust the number of slices choosed for automatic slicing, take note of the following points:

If the value of `slices` is equal to the number of shards in an index, you can obtain the optimal query performance. A value greater than the number of shards in an index, for example, 500, may not improve efficiency but increase overheads, resulting in performance degradation.

Update performance scales linearly across available resources with the number of slices.

Both the documents that are being reindexed and cluster resources determine whether query or update performance is dominant during the runtime.

8.2.2.4.7. Obtain a new property

Assume that you have created an index without dynamic mappings. Then, you add data to the index and add a mapping value to obtain more fields from the data. Sample code:

```
PUT test
{
 "mappings": {
  "_doc": {
   "dynamic": false,
   "properties": {
    "text": {"type": "text"}
   }
  }
 }
}
POST test/_doc? refresh
{
 "text": "words words",
 "flag": "bar"
}
POST test/_doc? refresh
{
 "text": "words words",
 "flag": "foo"
}
PUT test/_mapping/_doc
{
 "properties": {
  "text": {"type": "text"},
  "flag": {"type": "text", "analyzer": "keyword"}
 }
}
```

`"dynamic"` : This field is set to false, indicating that the new field will not be indexed and will only be stored in `_source` .

`PUT test/_mapping/_doc` : This code indicates that the mapping will be updated to add a new `flag` field. To add a new field, reindex all documents by using this field.

If you search for data records by using this field, no results will be returned.

```
POST test/_search? filter_path=hits.total
{
 "query": {
  "match": {
   "flag": "foo"
  }
 }
}
```

```
{
 "hits" : {
  "total" : 0
 }
}
```

However, you can initiate a `_update_by_query` request to add the new mapping:

```
POST test/_update_by_query? refresh&conflicts=proceed
POST test/_search? filter_path=hits.total
{
 "query": {
  "match": {
   "flag": "foo"
  }
 }
}
```

```
{
 "hits" : {
  "total" : 1
 }
}
```

You can perform the same operation if you want to add a field to a multifield.

8.2.2.5. Reindex

> **Note** The reindex operation does not attempt to specify the destination index. It does not copy the settings of the source index. To perform the `_reindex` operation, you must specify information about the destination index, including mappings, shard counts, and replicas.

In its basic form, the `_reindex` operation copies a document from one index to another. In the following example, documents are copied from the `twitter` index to the `new_twitter` index:

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

Sample response:

```
{
 "took" : 147,
 "timed_out": false,
 "created": 120,
 "updated": 0,
 "deleted": 0,
 "batches": 1,
 "version_conflicts": 0,
 "noops": 0,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": -1.0,
 "throttled_until_millis": 0,
 "total": 120,
 "failures" : [ ]
}
```

Similar to `_update_by_query`, `_reindex` obtains a snapshot of the source index. To avoid version conflicts, the destination index that is created based on the snapshot must be different from the source index. The `dest` element can be configured in a similar way as that in the index operation. This allows you to implement optimistic concurrency control. If you omit the `version_type` parameter or set it to `internal`, Elasticsearch dumps documents to the destination index. This will overwrite any documents that have the same type and ID:

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "version_type": "internal"
 }
}
```

If you set `version_type` to `external`, Elasticsearch retains `version` in the source document, creates missing documents, and updates any documents whose version in the destination index is earlier than its version in the source index.

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "version_type": "external"
 }
}
```

If you set `op_type` to `create`, Elasticsearch calls the `_reindex` operation. This allows you to create the missing documents in the destination index. In this case, all existing documents will cause version conflicts.

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "op_type": "create"
 }
}
```

Version conflicts automatically stop a `_reindex` process. However, if you set `"conflicts": "proceed"`, version conflicts are only counted and do not stop the _reindex process.

```
POST _reindex
{
 "conflicts": "proceed",
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "op_type": "create"
 }
}
```

You can add a type or query for `source` to limit the number of documents. This will only copy tweets made from `kimchy` to `new_twitter`.

```
POST _reindex
{
 "source": {
  "index": "twitter",
  "type": "_doc",
  "query": {
   "term": {
    "user": "kimchy"
   }
  }
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

The `index` and `type` fields in `source` can be lists. This allows you to copy multiple documents from a large amount of source code by using one request. After you specify the two fields, you can copy documents of the `_doc` and `post` types from the `twitter` and `blog` indexes. The copied documents will include the `post` type in the `twitter` index and the `_doc` type in the `blog` index. To configure more parameters for the operation, you can specify parameters in the `query` field.

The reindex operation does not handle ID conflicts. When ID conflicts occur, the destination index still remains valid. However, the iteration order is not well defined. In this way, you cannot predict which document is retained.

```
POST _reindex
{
 "source": {
  "index": ["twitter", "blog"],
  "type": ["_doc", "post"]
 },
 "dest": {
  "index": "all_together"
 }
}
```

You can specify the `size` field to limit the number of documents you want to process. If you set size to 1, only one document will be copied from `twitter` to `new_twitter` :

```
POST _reindex
{
 "size": 1,
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

You can use `sort` to copy a specific set of documents from the `twitter` index. Sorting makes the scroll less efficient but it is useful in some scenarios. We recommend that you use the `size` or `sort` field. In the following example, 10,000 documents are copied from `twitter` to `new_twitter`:

```
POST _reindex
{
 "size": 10000,
 "source": {
  "index": "twitter",
  "sort": { "date": "desc" }
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

The `source` section supports all the elements that are supported in a search request. For example, a subset of the fields from the original documents can be reindexed by using `source` filtering, as shown in the following code:

```
POST _reindex
{
 "source": {
  "index": "twitter",
  "_source": ["user", "_doc"]
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

Like `_update_by_query`, `_reindex` supports a script that is used to modify documents. Unlike `_update_by_query`, the script is allowed to modify the metadata in a document. The following code is used to raise the version of the source document:

```
POST _reindex
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter",
  "version_type": "external"
 },
 "script": {
  "source": "if (ctx._source.foo == 'bar') {ctx._version++; ctx._source.remove('foo')}",
  "lang": "painless"
 }
}
```

Similar to `_update_by_query`, the reindex operation allows you to set `ctx.op` to change the operation you want to perform on the destination index.

- `noop`:

  Set `ctx.op` to noop if the script field specifies that it is unnecessary to index the document in the destination index. This operation will be reported in the `noop` counter in the response body.

- `delete`:

  Set ctx.op to `delete` if the script field specifies that the document must be deleted from the destination index. The deletion operation will be reported in the `deleted` counter in the response body.

If you do not set `ctx.op` to noop or delete, an error will be returned. The same rule applies to other fields in `ctx`.

Use caution when you modify the following fields:

- `_id`
- `_type`
- `_index`
- `_version`
- `_routing`

If you set `_version` to `null` or clear this field from `ctx`, the version will not be included in an index request. This causes the document to be overwritten in the target index regardless of the target version or the version type you use in the `_reindex` request.

If `_reindex` discovers a document that contains routing, the routing will be retained unless it is changed by the script. In a `dest` request, you can set `routing` to the following values:

- `keep`

  Sets the routing in a bulk request sent for each match item to the routing for the match item. This is the default value.

- `discard`

  Sets the routing in a bulk request sent for each match item to `null`.

- `=<some text>`

  Sets the routing in a bulk request sent for each match item to all texts after `=`.

For example, you can use the following request to copy all documents from the `source` index with the company name `cat` to the `dest` index with routing set to `cat`:

```
POST _reindex
{
 "source": {
  "index": "source",
  "query": {
   "match": {
    "company": "cat"
   }
  }
 },
 "dest": {
  "index": "dest",
  "routing": "=cat"
 }
}
```

By default, `_reindex` uses 1,000 scroll batches. You can specify the `size` field in the `source` element to change the batch size.

```
POST _reindex
{
 "source": {
  "index": "source",
  "size": 100
 },
 "dest": {
  "index": "dest",
  "routing": "=cat"
 }
}
```

The reindex operation also allows you to use an ingest node by specifying `pipeline`, as shown in the following code.

```
POST _reindex
{
 "source": {
  "index": "source"
 },
 "dest": {
  "index": "dest",
  "pipeline": "some_ingest_pipeline"
 }
}
```

8.2.2.5.1. Reindex from a remote cluster

You can call the reindex operation to reindex from a remote Elasticsearch cluster.

```
POST _reindex
{
 "source": {
  "remote": {
   "host": "http://otherhost:9200",
   "username": "user",
   "password": "pass"
  },
  "index": "source",
  "query": {
   "match": {
    "test": "data"
   }
  }
 },
 "dest": {
  "index": "dest"
 }
}
```

The `host` parameter must contain a mode, host, and port number. A sample value is `https://otherhost:9200`). The `username` and `password` parameters are optional. If they are present, `_reindex` will connect to the remote Elasticsearch node by using basic authentication. When you use basic authentication, make sure that HTTPS is used. Otherwise, the password is sent in plaintext.

You must specify the `reindex.remote.whitelist` property to add the remote host to the whitelist in the elasticsearch.yml file. You can set the value to a comma-delimited list that contains combinations of allowed remote hosts and port numbers, for example, `otherhost:9200, another:9200, 127.0.10.*:9200, localhost:*` . The whitelist uses only the host and port number because the mode is omitted. Sample code:

```
reindex.remote.whitelist: "otherhost:9200, another:9200, 127.0.10.*:9200, localhost:*"
```

The whitelist must be configured on any nodes that are used to coordinate the reindex operation.

This feature applies to remote clusters deployed for Elasticsearch of any versions. This allows you to update Elasticsearch to the latest version by reindexing from a cluster of the source version.

To send queries to an early version of Elasticsearch, you can send the `query` parameter to the remote host without verification or modification.

> **Note** Reindexing from a remote cluster does not support manual slicing or automatic slicing.

Reindexing from a remote server uses an on-heap buffer with the default maximum size of 100 MB. If the remote index includes very large documents, you must specify a smaller batch size. In the following example, the batch size is set to `10` , which is very small.

```
POST _reindex
{
 "source": {
  "remote": {
   "host": "http://otherhost:9200"
  },
  "index": "source",
  "size": 10,
  "query": {
   "match": {
    "test": "data"
   }
  }
 },
 "dest": {
  "index": "dest"
 }
}
```

You can use the `socket_timeout` field to specify the socket read timeout period during remote connections. You can also use the `connect_timeout` field to specify the connection timeout period. The default values of the two fields are 30 seconds. In the following example, the socket_timeout field is set to 1m, and the connect_timeout field is set to 10s.

```
POST _reindex
{
 "source": {
  "remote": {
   "host": "http://otherhost:9200",
   "socket_timeout": "1m",
   "connect_timeout": "10s"
  },
  "index": "source",
  "query": {
   "match": {
    "test": "data"
   }
  }
 },
 "dest": {
  "index": "dest"
 }
}
```

8.2.2.5.2. URL parameters

In addition to standard parameters, such as `pretty`, the reindex operation supports the following parameters: `refresh`, `wait_for_completion`, `wait_for_active_shards`, `timeout`, `scroll`, and `requests_per_second`.

If the `refresh` parameter is specified in a request, Elasticsearch refreshes all indexes written by the request. This is different from the `refresh` parameter in an index operation. If the refresh parameter is specified in an index operation, Elasticsearch refreshes only shards that receive new data.

If a request contains `wait_for_completion=false`, Elasticsearch performs some prechecks and starts the request. Then, Elasticsearch returns a `task` that can be used together with the task operation to cancel the task or obtain the status of the task. Elasticsearch also creates a record of this task as a document in `.tasks/task/${taskId}`. You can retain or delete this document as needed. After the task is complete, delete the document so that Elasticsearch can reclaim the disk space occupied by the document.

The `wait_for_active_shards` parameter controls the number of replicas that must be active before Elasticsearch proceeds with a reindexing request. For more information, see Active shards. The `timeout` parameter specifies the period of time that a write request must wait for unavailable shards to become available. Both of the preceding parameters work the same way as they work in the bulk operation. Because `_reindex` uses scroll searches, you can specify the `scroll` parameter to control the period of time that is required to keep search contexts alive, such as `?scroll=10m`. The default value is 5m.

You can set the `requests_per_second` parameter to any positive decimal number, such as `1.4`, `6`, and `1000`, to throttle the rate at which `_reindex` issues batches of index operations. Each batch is padded with a wait time to throttle requests. You can set `requests_per_second` to `-1` to disable throttling.

Throttling is performed by waiting between batches so that `_reindex` can assign the internal `scroll` request a timeout period that takes the padding time into account. The padding time is the difference between the batch size divided by `requests_per_second` and the time that is required for a write operation. The default batch size is `1,000`. Therefore, if `requests_per_second` is set to `500`:

```
target_time = 1000 / 500 per second = 2 seconds
wait_time = target_time - write_time = 2 seconds - .5 seconds = 1.5 seconds
```

A batch is issued as a single `_bulk` request. If the size of a batch is large, Elasticsearch creates many requests and waits for a specific period of time before proceeding with the next batch. This is considered a burst operation instead of a smooth operation. The default value of requests_per_second is `-1`.

8.2.2.5.3. Response body

The following sample code provides a response in JSON format:

```
{
 "took": 639,
 "timed_out": false,
 "total": 5,
 "updated": 0,
 "created": 5,
 "deleted": 0,
 "batches": 1,
 "noops": 0,
 "version_conflicts": 2,
 "retries": {
  "bulk": 0,
  "search": 0
 },
 "throttled_millis": 0,
 "requests_per_second": 1,
 "throttled_until_millis": 0,
 "failures": [ ]
}
```

- `took`

  The total number of milliseconds that the operation took.
- `timed_out`

  If a request that is executed during reindexing times out, this field is set to `true`.
- `total`

  The number of documents that are processed.
- `updated`

  The number of documents that are updated.
- `created`

  The number of documents that are created.
- `deleted`

  The number of documents that are deleted.
- `batches`

  The number of scroll responses that are pulled back by the reindex operation.
- `noops`

The number of documents that are ignored because the script used for the reindex operation returns a `noop` value for `ctx.op`.

- `version_conflicts`

  The number of version conflicts incurred by the reindex operation.

- `retries`

  The number of retries attempted by the reindex operation. `bulk` indicates the number of bulk operations retried and `search` indicates the number of search operations retried.

- `throttled_millis`

  The number of milliseconds the request remains in the sleep state to conform to `requests_per_second`.

- `requests_per_second`

  The number of requests per second that are successfully executed during reindexing.

- `throttled_until_millis`

  The value of this field is always 0 in the response of the `_reindex` operation. This field is valid only when the task operation is called. In this case, this field indicates when (in milliseconds from epoch) the next throttling request will be executed to conform to `requests_per_second`.

- `failures`

  If unrecoverable errors occur during reindexing, an array of failures is returned. If this field is not empty, the request is stopped because of these failures. The reindex operation is implemented by using batches. Failures can cause the operation to stop, and all failures in the current batch are collected into the array. You can use the `conflicts` option to prevent the reindex operation from stopping because of version conflicts.

8.2.2.5.4. Obtain the status of all running reindex requests

You can call the task operation to obtain the status of all running reindex requests.

```
GET _tasks?detailed=true&actions=*reindex
```

Sample response:

```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
   "tasks" : {
    "r1A2WoRbTwKZ516z6NEs5A:36619" : {
     "node" : "r1A2WoRbTwKZ516z6NEs5A",
     "id" : 36619,
     "type" : "transport",
     "action" : "indices:data/write/reindex",
     "status" : {
      "total" : 6154,
      "updated" : 3500,
      "created" : 0,
      "deleted" : 0,
      "batches" : 4,
      "version_conflicts" : 0,
      "noops" : 0,
      "retries": {
       "bulk": 0,
       "search": 0
      },
      "throttled_millis": 0
     },
     "description" : ""
    }
   }
  }
 }
}
```

The `status` object indicates the actual status. It is the same as a response in JSON format, except the `total` field. The `total` field indicates the total number of operations performed by `_reindex`. You can add the `updated`, `created`, and `deleted` fields to estimate the progress. When the sum of the values for the updated, created, and deleted fields are equal to the value of the `total` field, the request is terminated.

You can specify the ID of a task to search for it. Example:

```
GET /_tasks/taskId:1
```

The task operation has the following advantages: It integrates with `wait_for_completion=false` and returns the status of completed tasks in a transparent manner. If tasks are complete and `wait_for_completion=false` is specified, the operation returns a `results` or `error` field. Note that `wait_for_completion=false` creates a document in `.tasks/task/${taskId}`. You can delete this document as needed.

8.2.2.5.5. Cancel a reindexing task

You can call the task operation to cancel a reindexing task.

```
POST _tasks/task_id:1/_cancel
```

You can also call the task operation to obtain `task_id`.

The cancellation may require a few seconds. The task operation continues to return tasks until it is awakened to cancel itself.

8.2.2.5.6. Rethrottling

You can call the `_rethrottle` operation to change the value of `requests_per_second` for a running reindexing task.

```
POST _reindex/task_id:1/_rethrottle?requests_per_second=-1
```

You can also call the task operation to obtain `task_id`.

`requests_per_second` can be set to `-1`, which is the same as the setting for the reindex operation, to disable throttling. Alternatively, it can be set to any decimal number such as `1.7` or `12` to throttle the number of requests per second. Rethrottling that speeds up a query takes effect immediately. However, rethrottling that slows down a query takes effect after the current batch is processed. This prevents scroll timeouts.

8.2.2.5.7. Perform reindexing to change the name of a field

You can call the `_reindex` operation to create a copy for an index and specify fields whose names you want to change. For example, you run the following command to create a document in an index:

```
POST test/_doc/1?refresh
{
 "text": "words words",
 "flag": "foo"
}
```

After the document is created, you want to change the field name `flag` to `tag`. You can call the `_reindex` operation to create another index.

```
POST _reindex
{
 "source": {
  "index": "test"
 },
 "dest": {
  "index": "test2"
 },
 "script": {
  "source": "ctx._source.tag = ctx._source.remove(\"flag\")"
 }
}
```

Run the following command to obtain the new document:

```
GET test2/_doc/1
```

The following result is returned:

```
{
 "found": true,
 "_id": "1",
 "_index": "test2",
 "_type": "_doc",
 "_version": 1,
 "_source": {
  "text": "words words",
  "tag": "foo"
 }
}
```

8.2.2.5.8. Slicing

The reindex operation allows you to perform reindexing in parallel with sliced scroll. During reindexing, you can split a request into slices. This improves the efficiency of the request.

Manual slicing

You can specify slice IDs and the total number of slices for each reindex request to manually split the request.

```
POST _reindex
{
 "source": {
  "index": "twitter",
  "slice": {
   "id": 0,
   "max": 2
  }
 },
 "dest": {
  "index": "new_twitter"
 }
}
POST _reindex
{
 "source": {
  "index": "twitter",
  "slice": {
   "id": 1,
   "max": 2
  }
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

You can run the following command to check whether the manual slicing succeeded:

```
GET _refresh
        POST new_twitter/_search?size=0&filter_path=hits.total
```

A result similar to the following code is returned. The code contains the `total` field with a reasonable value.

```
{
 "hits": {
  "total": 120
 }
}
```

Automatic slicing

You can also perform sliced scroll to split `_uid` so that the parallel `_reindex` operation can be automatically performed. Set the `slices` parameter to specify the number of slices to use.

```
POST _reindex?slices=5&refresh
{
 "source": {
  "index": "twitter"
 },
 "dest": {
  "index": "new_twitter"
 }
}
```

You can run the following command to check whether the automatic slicing succeeded:

```
POST new_twitter/_search?size=0&filter_path=hits.total
```

A result similar to the following code is returned. The code contains the `total` field with a reasonable value.

```
{
 "hits": {
  "total": 120
 }
}
```

If the `slices` parameter is set to `auto`, Elasticsearch determines the number of slices to use. In this case, each shard uses one slice until the number of slices reaches the upper limit. If multiple source indexes exist, Elasticsearch sets the number of slices based on the index that has the least number of shards.

If you specify the `slices` parameter when you call the `_reindex` operation, manual slicing becomes automated. This generates subrequests that have different characteristics.

- You can call the task operation to view the subrequests. These subrequests are subtasks of request tasks that contain `slices`.
- When you query the status of request tasks that contain `slices`, only the status of the tasks that have completed slicing can be obtained.
- Subrequests can be separately processed for operations such as cancellation and rethrottling.
- If you use `slices` to rethrottle a request, only unfinished subrequests are proportionally rethrottled.
- If you use `slices` to cancel a request, all its subrequests are canceled.

- When `slices` is used, documents are not evenly allocated to subrequests. All documents are processed, but some slices may be greater than other slices. Greater slices may have a more even distribution.
- In requests that contain `slices`, the `requests_per_second` and `size` parameters are proportionally allocated to subrequests. Because the distribution is uneven, a request that contains `slices` and a specified `size` may not return a reindexed document that has the specified `size` value.
- The source index snapshots obtained by each subrequest are slightly different, but these snapshots are created at almost the same time.

Set the number of slices

For automatic slicing in which `slices` is set to `auto`, Elasticsearch sets a reasonable number of slices for most indexes. If you want to perform manual slicing or adjust automatic slicing, use the following guidelines:

If the value of `slices` is equal to the number of shards in an index, you can obtain the optimal query performance. If an index has a large number of shards, such as 500, set `slices` to a small number. Too many slices affect performance. Setting `slices` to a value greater than the number of shards does not improve efficiency but increases overheads.

Reindexing performance scales linearly across available resources based on the number of slices.

The documents that are being reindexed and cluster resources determine whether query or reindexing performance is dominant during the runtime.

8.2.2.5.9. Reindex daily indexes

You can use `_reindex` in combination with painless to reindex daily indexes and apply a new template to existing documents.

For example, your index consists of the following documents:

```
PUT metricbeat-2016.05.30/_doc/1?refresh
{"system.cpu.idle.pct": 0.908}
PUT metricbeat-2016.05.31/_doc/1?refresh
{"system.cpu.idle.pct": 0.105}
```

The new template of the `metricbeat-*` index is loaded to Elasticsearch, but this template applies only to newly created indexes. The painless operation can be used to reindex existing documents and apply new templates.

You can use the following script to extract the date from an index name and create an index with `-1` appended. All data from `metricbeat-2016.05.31` are reindexed into `metricbeat-2016.05.31-1`.

```
POST _reindex
{
 "source": {
  "index": "metricbeat-*"
 },
 "dest": {
  "index": "metricbeat"
 },
 "script": {
  "lang": "painless",
  "source": "ctx._index = 'metricbeat-' + (ctx._index.substring('metricbeat-'.length(), ctx._index.length())) + '-1'"
 }
}
```

After the script is executed, you can find all documents from the metricbeat index in the `*-1` index.

```
GET metricbeat-2016.05.30-1/_doc/1
GET metricbeat-2016.05.31-1/_doc/1
```

You can use the preceding script in combination with the method described in Perform reindexing to change the name of a field to add existing data to the new index and rename fields as needed.

8.2.2.5.10. Extract a random subset of an index

`_reindex` can be used to extract a random subset of an index for testing.

```
POST _reindex
{
 "size": 10,
 "source": {
  "index": "twitter",
  "query": {
   "function_score" : {
    "query" : { "match_all": {} },
    "random_score" : {}
   }
  },
  "sort": "_score"
 },
 "dest": {
  "index": "random_twitter"
 }
}
```

`sort` : The default value is `_doc` . If you want to sort documents by `_score` , you must change the default value to `_score` . Otherwise, `random_score` is invalid.