ALIBABA CLOUD

阿里云

Blink

Blink Exclusive Mode (Phased-Out for Alibaba Cloud)

Document Version: 20231114

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
⚠ **Warning**	You can use this note type to provide information of which the user must take special notice to prevent risks or service interruptions, such as precautions and prerequisites.	⚠ **Warning** A restart operation may cause a temporary service interruption. We recommend that you perform a restart during off-peak hours or after data is backed up. You can also contact Alibaba Cloud technical support.
ⓘ **Important**	You can use this note type to provide information that does not involve risks but is necessary to know for an operation, such as subsequent operations and limitations.	ⓘ **Important** When you log on to the system again, you must set a new password for your logon account.
? **Note**	You can use this note type to provide information that is simply good-to-know, such as best practices and tips.	? **Note** You can press 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.Release notes ---- 17
1.1. Blink-3.7.10 ---- 17
1.2. Blink 3.7.9 ---- 17
1.3. Blink-3.7.7 ---- 17
1.4. Blink-3.6.8 ---- 18
1.5. Blink-3.6.5 ---- 18
1.6. Blink 3.6.2 ---- 19
1.7. Blink-3.6.0 ---- 19
1.8. Blink-3.5.0-hotfix ---- 20
1.9. Blink 3.4.4 ---- 20
1.10. Blink-3.4.3 ---- 20
1.11. Blink 3.3.0 ---- 23
1.12. Blink 3.2.3 ---- 24
1.13. Blink 3.2.1 ---- 25
1.13.1. Blink 3.2.1 release notes ---- 25
1.13.2. API compatibility report: Blink 3.2 and Flink 1.5.1 ---- 26
1.13.3. Incompatible SQL items: Blink 3.0 and Blink 2.0 ---- 28
2.Product Introduction ---- 30
2.1. Overview ---- 30
2.2. Development history ---- 34
2.3. Workflow ---- 35
2.4. Upstream and downstream data stores ---- 37
2.5. Product security ---- 38
2.6. Limits ---- 39
2.7. Architecture of Realtime Compute for Apache Flink in e... ---- 40
3.Pricing ---- 43

3.1. Billing unit 43
3.2. Billing methods 43
3.3. Specification selection 45
3.4. Renewal 47
3.4.1. Manual renewal 47
3.4.2. Auto-renewal 47
3.5. Change resource configurations 48
4.Preparation 51
4.1. Grant permissions to a RAM user 51
4.2. Activate Realtime Compute for Apache Flink and create... 52
4.3. Role authorization 56
4.3.1. Assign a RAM role to an account that uses Realtime... 56
5.Blink SQL reference 61
5.1. Overview 61
5.2. Keywords 61
5.3. Basic concepts 63
5.3.1. Time zone 63
5.3.2. Time attributes 65
5.3.3. Watermark 68
5.3.4. Computed column 69
5.4. Data types 70
5.4.1. Data type conversion 70
5.4.2. Mathematical and logical operations between data t... 72
5.5. Create a data view 73
5.6. DDL statements 75
5.6.1. Overview 75
5.6.2. Create a source table 78
5.6.2.1. Overview of source tables 78

5.6.2.2. Create an Oracle database source table ---- 81
5.6.2.3. Create a Hologres source table ---- 85
5.6.2.4. Create a Log Service source table ---- 88
5.6.2.5. Create a source table ---- 94
5.6.2.6. Create a Message Queue for Apache Kafka sourc.. ---- 100
5.6.2.7. Create a Tablestore source table ---- 117
5.6.2.8. Create a full MaxCompute source table ---- 120
5.6.2.9. Create an incremental MaxCompute source table ---- 127
5.6.3. Create a result table ---- 133
5.6.3.1. Overview of result tables ---- 133
5.6.3.2. Create an Oracle database result table ---- 133
5.6.3.3. Create a Hologres result table ---- 136
5.6.3.4. Create an AnalyticDB for MySQL V2.0 result table ---- 142
5.6.3.5. Create a Log Service result table ---- 145
5.6.3.6. Create a result table ---- 148
5.6.3.7. Create a Tablestore result table ---- 153
5.6.3.8. Create an ApsaraDB RDS result table ---- 155
5.6.3.9. Create a MaxCompute result table ---- 161
5.6.3.10. Create an ApsaraDB for HBase result table ---- 168
5.6.3.11. Create an Elasticsearch result table ---- 173
5.6.3.12. Create a TSDB result table ---- 177
5.6.3.13. Create a Message Queue for Apache Kafka res... ---- 180
5.6.3.14. Create a HybridDB for MySQL result table ---- 183
5.6.3.15. Create an ApsaraDB RDS for SQL Server result... ---- 185
5.6.3.16. Create an ApsaraDB for Redis result table ---- 191
5.6.3.17. Create an ApsaraDB for MongoDB result table ---- 195
5.6.3.18. Create an AnalyticDB for MySQL V3.0 result tab.. ---- 196
5.6.3.19. Create a custom result table ---- 199

5.6.3.20. Create a Phoenix5 result table 205
5.6.3.21. Create an AnalyticDB for PostgreSQL result table 207
5.6.3.22. Create an InfluxDB result table 210
5.6.4. Create a dimension table 212
5.6.4.1. Overview 212
5.6.4.2. Create a Hologres dimension table 214
5.6.4.3. Create a Tablestore dimension table 218
5.6.4.4. Create an ApsaraDB RDS for MySQL dimension t... 221
5.6.4.5. Create an ApsaraDB for HBase dimension table 227
5.6.4.6. Create a MaxCompute dimension table 234
5.6.4.7. Create an ApsaraDB for Redis dimension table 242
5.6.4.8. Create an Elasticsearch dimension table 245
5.6.4.9. Create a Phoenix5 dimension table 247
5.6.4.10. Create an AnalyticDB for MySQL V3.0 dimensio... 251
5.6.4.11. Create an Oracle database dimension table 256
5.7. DML statement 259
5.7.1. EMIT statements 259
5.7.2. INSERT INTO statements 263
5.8. Query statements 264
5.8.1. SELECT statements 264
5.8.2. WHERE 266
5.8.3. HAVING statement 267
5.8.4. GROUP BY statement 268
5.8.5. JOIN statements 268
5.8.6. JOIN statements for dimension tables 271
5.8.7. IntervalJoin statement 273
5.8.8. UNION ALL 278
5.8.9. TopN 279

5.8.10. GROUPING SETS clause 285
5.8.11. CEP statements 286
5.8.12. Deduplication statements 293
5.9. Window functions 295
5.9.1. Overview 295
5.9.2. TUMBLE 297
5.9.3. HOP 301
5.9.4. SESSION 304
5.9.5. OVER windows 306
5.10. Built-in functions 314
5.10.1. String functions 314
5.10.1.1. REGEXP_EXTRACT 314
5.10.1.2. REGEXP_REPLACE 316
5.10.1.3. REPEAT 317
5.10.1.4. REPLACE 318
5.10.1.5. REVERSE 319
5.10.1.6. RPAD 319
5.10.1.7. SPLIT_INDEX 321
5.10.1.8. STR_TO_MAP 322
5.10.1.9. SUBSTRING 323
5.10.1.10. TO_BASE64 324
5.10.1.11. TRIM 325
5.10.1.12. UPPER 325
5.10.1.13. CHAR_LENGTH 326
5.10.1.14. CHR 327
5.10.1.15. CONCAT 328
5.10.1.16. CONCAT_WS 329
5.10.1.17. FROM_BASE64 330

5.10.1.18. HASH_CODE 330
5.10.1.19. INITCAP 331
5.10.1.20. INSTR 332
5.10.1.21. JSON_VALUE 333
5.10.1.22. KEYVALUE 334
5.10.1.23. LOWER 335
5.10.1.24. LPAD 336
5.10.1.25. MD5 338
5.10.1.26. OVERLAY 338
5.10.1.27. PARSE_URL 339
5.10.1.28. POSITION 340
5.10.1.29. REGEXP 341
5.10.2. Mathematical functions 342
5.10.2.1. Addition 342
5.10.2.2. Subtraction 343
5.10.2.3. Multiplication 343
5.10.2.4. Division 344
5.10.2.5. ABS 345
5.10.2.6. ACOS 345
5.10.2.7. BIN 346
5.10.2.8. ASIN 347
5.10.2.9. ATAN 348
5.10.2.10. BITAND 349
5.10.2.11. BITNOT 349
5.10.2.12. BITOR 350
5.10.2.13. BITXOR 351
5.10.2.14. CARDINALITY 352
5.10.2.15. CONV 352

5.10.2.16. COS 353
5.10.2.17. COT 354
5.10.2.18. EXP 355
5.10.2.19. E 356
5.10.2.20. FLOOR 356
5.10.2.21. LN 357
5.10.2.22. LOG 358
5.10.2.23. LOG10 359
5.10.2.24. LOG2 360
5.10.2.25. PI 360
5.10.2.26. POWER 361
5.10.2.27. RAND 362
5.10.2.28. SIN 362
5.10.2.29. SQRT 363
5.10.2.30. TAN 364
5.10.2.31. CEIL 365
5.10.2.32. CHARACTER_LENGTH 365
5.10.2.33. DEGREES 366
5.10.2.34. MOD 367
5.10.2.35. ROUND 367
5.10.3. Date functions 369
5.10.3.1. LOCALTIMESTAMP 369
5.10.3.2. CURRENT_DATE 369
5.10.3.3. CURRENT_TIMESTAMP 370
5.10.3.4. DATEDIFF 370
5.10.3.5. DATE_ADD 371
5.10.3.6. DATE_FORMAT 372
5.10.3.7. DATE_SUB 373

5.10.3.8. DAYOFMONTH 374
5.10.3.9. EXTRACT 375
5.10.3.10. FROM_UNIXTIME 376
5.10.3.11. HOUR 377
5.10.3.12. LOCALTIME 378
5.10.3.13. MINUTE 378
5.10.3.14. MONTH 379
5.10.3.15. NOW 380
5.10.3.16. SECOND 381
5.10.3.17. TIMESTAMPADD 382
5.10.3.18. TO_DATE 383
5.10.3.19. TO_TIMESTAMP 384
5.10.3.20. UNIX_TIMESTAMP 385
5.10.3.21. WEEK 386
5.10.3.22. YEAR 387
5.10.4. Logical functions 388
5.10.4.1. = 388
5.10.4.2. > 389
5.10.4.3. >= 390
5.10.4.4. <= 390
5.10.4.5. < 391
5.10.4.6. <> 392
5.10.4.7. AND 393
5.10.4.8. BETWEEN AND 393
5.10.4.9. IS NOT FALSE 395
5.10.4.10. IS NOT NULL 396
5.10.4.11. IS NOT TRUE 397
5.10.4.12. IS NOT UNKNOWN 397

5.10.4.13. IS NULL 398
5.10.4.14. IS TRUE 399
5.10.4.15. IS UNKNOWN 400
5.10.4.16. LIKE 401
5.10.4.17. NOT 402
5.10.4.18. NOT BETWEEN AND 403
5.10.4.19. IN 405
5.10.4.20. OR 405
5.10.4.21. IS DISTINCT FROM 406
5.10.4.22. IS NOT DISTINCT FROM 407
5.10.4.23. NOT IN 408
5.10.5. Conditional functions 409
5.10.5.1. CASE WHEN 409
5.10.5.2. COALESCE 410
5.10.5.3. IF 411
5.10.5.4. IS_ALPHA 412
5.10.5.5. IS_DECIMAL 413
5.10.5.6. IS_DIGIT 414
5.10.5.7. NULLIF 415
5.10.6. Table-valued functions 416
5.10.6.1. GENERATE_SERIES 416
5.10.6.2. JSON_TUPLE 417
5.10.6.3. STRING_SPLIT 418
5.10.6.4. MULTI_KEYVALUE 419
5.10.7. Type conversion function 420
5.10.7.1. CAST 420
5.10.8. Aggregate functions 421
5.10.8.1. AVG 421

5.10.8.2. CONCAT_AGG 422
5.10.8.3. COUNT 423
5.10.8.4. FIRST_VALUE 424
5.10.8.5. LAST_VALUE 425
5.10.8.6. MAX 428
5.10.8.7. MIN 429
5.10.8.8. SUM 429
5.10.8.9. VAR_POP 430
5.10.8.10. STDDEV_POP 431
5.10.9. Other functions 432
5.10.9.1. UUID 432
5.10.9.2. DISTINCT 433
5.11. UDXs 436
5.11.1. Overview 436
5.11.2. UDF 440
5.11.3. UDAF 443
5.11.4. UDTF 448
5.11.5. Develop a UDX by using IntelliJ IDEA 453
6.Blink SQL Development Guide 458
6.1. Overview 458
6.2. Data storage 458
6.2.1. Overview 458
6.2.2. Data storage resource registration 462
6.2.2.1. Register an AnalyticDB for MySQL instance 462
6.2.2.2. Register a Tablestore instance 463
6.2.2.3. Register an ApsaraDB for RDS instance 464
6.2.2.4. Register a Log Service project 465
6.2.3. Configure a whitelist for accessing storage resources 467

6.3. Job development 467
6.3.1. Develop a job 467
6.3.2. Publish a job 471
6.3.3. Start a job 471
6.3.4. Suspend a job 472
6.3.5. Terminate a job 472
6.4. Job debugging 473
6.4.1. Perform local debugging 473
6.4.2. Online debugging 477
6.5. Job administration 479
6.5.1. Go to the Job Administration page 479
6.5.2. Overview 479
6.5.3. Metrics 483
6.5.4. Timeline 488
6.5.5. Failover 488
6.5.6. Checkpoints 489
6.5.7. JobManager 490
6.5.8. TaskExecutor 491
6.5.9. Data lineage 492
6.5.10. Properties and parameters 494
6.5.11. Job diagnosis 495
6.6. Job optimization 495
6.6.1. Overview 495
6.6.2. Recommended Flink SQL practices 496
6.6.3. Performance optimization by using automatic configu.. 505
6.6.4. Performance optimization by using auto scaling 513
6.6.5. Optimize performance by manual configuration 518
6.6.6. Typical backpressure scenarios and optimization idea... 526

6.6.7. SQL Tuning Advisor 529
6.6.7.1. Partitioned All Cache 530
6.6.7.2. miniBatch and microBatch 531
6.6.7.3. Cache policy 531
6.6.7.4. Asynchronous mode 534
6.6.7.5. APPROX_COUNT_DISTINCT 537
6.6.7.6. Local-global optimization 538
6.6.7.7. ROW_NUMBER OVER WINDOW 539
6.6.7.8. Partial-final optimization 541
6.7. Monitoring and alerting 542
6.8. Customize log levels and download paths 543
6.9. Manage Blink versions of a Realtime Compute for Apac... 546
6.10. Monitoring and alerting 549
7.Blink Datastream Development Guide 550
7.1. Overview 550
7.2. Configure a whitelist for accessing storage resources 551
7.3. Set custom parameters 551
7.4. Monitoring 552
7.5. Develop a job 553
7.6. Publish a job 555
7.7. Develop a job 558
7.8. Example of DataStream jobs 558
7.8.1. Read data from DataHub 558
7.8.2. Read data from Message Queue for Apache Kafka 562
7.8.3. Read data from DataHub and write data to ApsaraD... 564
7.8.4. Read data from Log Service 568
8.Best Practices 571
8.1. Best practices of Realtime Compute in the e-commerce... 571

8.1.1. Real-time PV and UV curves in e-commerce scenario.. 571
8.1.2. Marketing coupons in e-commerce scenarios 576
8.1.3. Real-time situation awareness and geographic distrib... 582
8.1.4. Latest transaction records in e-commerce scenarios 585
8.2. Best practices of Realtime Compute in the live streamin.. 588
8.2.1. Core video metric monitoring for live streaming 588
8.2.2. Digital operations for live streaming 593
9.Agreements 599
9.1. Realtime Compute for Apache Flink Service Level Agree... 599

1.Release notes

1.1. Blink-3.7.10

This topic describes major updates and bug fixes in Blink 3.7.10.

Major updates

The underlying client dependency of Message Queue for Apache RocketMQ connectors is upgraded. Therefore, you must change the value of the endpoint parameter in all the DDL statements in which the type parameter is set to mq to the TCP internal endpoint of Message Queue for Apache RocketMQ. For more information, see Announcement on the settings of internal TCP endpoints.

Major bug fixes

Fixes the defect of AutoScale.

1.2. Blink 3.7.9

This topic describes the major features and bug fixes in Blink 3.7.9.

Major features

Partitioned join is supported for dimension tables.

> **Note**
> - If the upstream data is Update Stream, the unique key of the upstream must contain the shuffle key when you perform a common partitioned join operation. This ensures the correctness of semantics.
> - If you enable partitioned join for dimension tables, hot spot issues may occur. Bucket partitioned join is introduced to address this issue. However, you cannot perform the bucket partitioned join operation if the upstream data is Update Stream. During the bucket partitioned join operation, the source table calculates and determines the concurrency for each data record and spreads each data record to a bucket in the downstream. Meanwhile, the dimension table caches each data record in the bucket to scatter hot spot data.

Major bug fixes

Fixes the bug that the checkpoint file is deleted by mistake because FileSegmentStateHandle is not registered with GeminiKeyedStateHandle.

1.3. Blink-3.7.7

This topic describes the major features and bug fixes in Blink 3.7.7.

Major bug fixes

- Fixes the bug that causes job checkpoints to occupy excessively large storage space.
- Fixes the bug that causes the residual file cleanup mechanism to delete checkpoint files by mistake.
- Fixes the bug that causes Blink to be unable to use Hologres physical tables with schemas.

1.4. Blink-3.6.8

This topic lists the major improvements and bug fixes in Blink 3.6.8.

Major improvements

- Improves the performance of writing data to Time Series Database (TSDB).
- Improves the performance of writing data to AnalyticDB for PostgreSQL.
- Sets the connection timeout period of Tablestore to 20s.

Major bug fixes

- Fixes the bug in which a failed job cannot resume during auto scaling.
- Fixes the bug that causes the NullPointerException error in a result table of AnalyticDB for MySQL V2.0.
- Fixes the bug that causes the ArrayIndexOutOfBoundsException error in jobs that use a RetracRank operator.
- Fixes the bug that causes data exceptions when a job that uses a RetracRank operator constantly receives the same retract message.
- Fixes the bug in which data is lost when you use Tunnel Service to read incremental data from a Tablestore source table after a full read.

1.5. Blink-3.6.5

This topic describes major updates and bug fixes in Blink 3.6.5.

Major updates

- The CPU utilization of DataHub source tables is reduced.
- The partition join feature of dimension tables is improved, and data can be loaded to partitions in dimension tables for which the cache parameter is set to ALL. This version supports parallel asynchronous optimization for multiple dimension tables. To enable the partition join feature for dimension tables that support parallel asynchronous optimization, you must set the **partitionedJoin** parameter to true in the WITH clause.
- The **startupMode** parameter is added to Log Service source tables. The startupMode parameter has the following valid values:
  - **TIMESTAMP**: indicates that data of a shard is consumed from the specified point of time.
  - **Earliest**: indicates that data of a shard is consumed from the earliest data in the shard.
  - **Latest**: indicates that data of a shard is consumed from the latest data in the shard.
  - **Group_Offsets**: indicates that data of a shard is consumed from a checkpoint that is stored in a specific consumer group.

  > **Note** The TIMESTAMP, Earliest, Latest, and Group_Offsets values take effect only when no checkpoint exists in the state data.
- The memory consumed to store vertex topology information is reduced for jobs that involve a large amount of data.
- The **timeFieldType** parameter in Oracle source tables supports multiple time formats. This parameter has the following valid values:
  - **TO_DATE**: the DATE type.
  - **TIMESTAMP**: the TIMESTAMP type.
  - **VARCHAR**: the DATETIME string type.
  - **NUMBER**: the NUMERIC type.

- The OracleSourceQueryCondition interface is supported and a class name is configured for the interface.

Major bug fixes

- Fixes the bug that data errors occur in MaxCompute dimension tables if a TaskManager uses multiple threads and the partition join feature is enabled.
- Fixes the bug that an error message is reported when the DDL field type of the sink node is different from the type of the inserted data.
- Fixes the bug that Realtime Compute for Apache Flink does not consume data in Log Service source tables.

1.6. Blink 3.6.2

This topic lists the major features and bug fixes in Blink 3.6.2.

Major features

- Optimizes the auto scaling feature so that scale-down operations are triggered for jobs with no traffic. This reduces resource consumption.
- Optimizes resource rescaling.
- Optimizes data parsing of the Log Service connector. The FastLogGroup method is used to parse data.
- Reduces the memory consumed to store vertex topology information for jobs that involve a large amount of data.

Major bug fixes

- Fixes the bug that Blink cannot consume the accumulated data in a Tablestore source table.
- Fixes the bug that NullPointException occurs when the cache parameter is set to ALL for a dimension table.
- Fixes the bug that a MaxCompute sink cannot submit data to a dynamic partition.

1.7. Blink-3.6.0

This topic describes major updates and bug fixes in Blink 3.6.0.

Major updates

- The following connectors are added: AnalyticDB for PostgreSQL, Phoenix5, and Hologres. For more information, see the following topics:
  - Create an AnalyticDB for PostgreSQL result table
  - Create a Phoenix5 dimension table
  - Create a Hologres source table
  - Create a Hologres dimension table
  - Create a Hologres result table
- The following features are added:
  - Keys can be written to Message Queue result tables in the CSV format, but cannot be written to Message Queue result tables in the binary format.
  - Gemini 2.0 (Gemini enhanced edition) is supported.

Major bug fixes

The bugs that cause the following issues are fixed:

- The following syntax error of a Kafka08 result table is reported: `Kafka dont have sufficient arguments`.
- A failover occurs because the node of a Tablestore source table is running online but a partition of the source table does not have data. Error: `java.lang.NullPointerException`.
- A failover occurs because a Tablestore dimension table is referenced by using registered storage resources but the token expires. Error: `OTSNoPermissionAccess, [Message]:You have no permission to access the requested resource, please contact the resource owner`.

1.8. Blink-3.5.0-hotfix

This topic describes major updates and bug fixes in Blink 3.5.0.

Major updates

- You are allowed to create an InfluxDB result table. For more information, see Create an InfluxDB result table.
- You are allowed to create a Phoenix5 result table. For more information, see Create a Phoenix5 result table.
- The SDK version of DataHub connectors is updated.

Major bug fixes

- Fixes the bug that no error message is returned when you use a global ORDER BY clause.
- Fixes the bug that a behavior-related error message is returned when you implicitly convert the VARCHAR data type at the underlying layer into an integer data type.
- Fixes the bug that exceptions occur in computing results when data is aggregated based on the time due to a calcite bug.
- Fixes the bug that the error message of `java.lang.NoClassDefFoundError: com/aliyun/datahub/client/auth/Account` is reported when you check the syntax of DataHub source tables.

1.9. Blink 3.4.4

This topic lists the major features and bug fixes in Blink 3.4.4.

Major improvements

Blink 3.4.4 supports reading data from the new partitions of MaxCompute source tables. You can specify the **subscribeNewPartition** parameter in the DDL statement to enable or disable this feature. If the subscribeNewPartition parameter is set to true and the partition parameter is not specified, the system constantly reads data from the new partitions of MaxCompute source tables. For more information, see Create a full MaxCompute source table.

Major bug fixes

Fixes ApsaraDB RDS connection issues. Blink uses the Druid database connection pool to connect to ApsaraDB RDS. If the connection is not used for a long period of time, job failovers may occur. To avoid these issues, Blink 3.4.4 optimizes the method of connecting the Druid connection pool to ApsaraDB RDS.

1.10. Blink-3.4.3

This topic describes the major features and bug fixes in Blink 3.4.3.

Major features

GeminiStateBackend is the new generation of a backend platform that uses GeminiDB. GeminiDB is a storage engine developed by Alibaba Cloud. The performance of GeminiStateBackend is 1.5 times that of NiagaraStateBackend. This performance rating is based on tests that are performed on running jobs. GeminiStateBackend has the following major benefits:

- Uses LSM-based indexing and hash indexing. LSM refers to log-structured merge-tree. LSM is adopted to improve write performance, and hash indexes are stored in memory to optimize LSM read amplification. Specifically, GeminiDB divides each file into different pages, and flushes and compresses data by page. Hash indexes are used to quickly locate the page where the data is stored based on keys. This way, the number of I/O operations is reduced and read performance is improved.
- Optimizes the cache policy. GeminiDB caches important information in memory, such as newly inserted data and compressed data that includes hotspots. For traditional LSM-based storage, the data is first flushed to disks. New data is cached after at least one read I/O operation is performed. This process reduces the cache hit ratio.
- Optimizes the policy of flushing data to disks. GeminiDB flushes data to disks only after the cached data occupies all the memory space. Therefore, no data files are generated if the memory space is sufficient and data is compressed in a timely manner. For traditional LSM-based storage, data is flushed to disks for persistence. If Blink is used, this process is no longer required. Blink provides the checkpointing mechanism to ensure data consistency, and data can be persisted when checkpoints are created.
- Supports in-memory compaction. The data records that reside in memory are relocated in a timely manner to maximize the available space. This allows you to optimize write amplification and reduce read I/O operations.
- Eliminates the Java Native Interface (JNI) overhead of RocksDB or Niagara by using Java.
- Supports incremental checkpointing.
- Supports the local recovery feature, which enables quick recovery after a job fails.
- Supports separation of computing from storage, which enables quick recovery after a job is restarted or rescaled. This feature is continuously optimized to improve user experience.

GeminiStateBackend requires the following configurations for DataStream and SQL jobs:

- DataStream jobs
  - API configuration

```
StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
GeminiStateBackend stateBackend = new GeminiStateBackend(checkpointDir);
// Configuration for gemini
Configuration config = new Configuration();
config.setString("state.backend.gemini.heap.size", "1024mb");
// set configuration to backend
stateBackend.setConfiguration(conf);
// use gemini as state backend
env.setStateBackend(new GeminiStateBackend(checkpointDir));
```

- Parameters

Parameter	Data type	Unit	Default value	Description
state.backend.gemini.ttl.ms	LONG	ms	-1 (This value indicates that this feature is disabled by default.)	Optional. The data retention period.
state.backend.gemini.heap.size	STRING	The following units are supported: ▪ 1024 ▪ 1024kb ▪ 1024mb ▪ 1024gb	No default value	Optional. The memory size that can be used for a single GeminiDB database. **Note** We recommend that you specify this parameter. If you do not specify the parameter, the backend calculates the default value based on the Java Virtual Machine (JVM) and TaskManager configurations.

- SQL jobs

```
# Use GeminiStateBackend as the backend.
state.backend.type=gemini
# Set the time to live (TTL) of the state data.
state.backend.gemini.ttl.ms=129600000
# Set the memory size that can be used for a single GeminiDB database. The unit is MB
. Note that the memory resources of operators must include the memory size that can b
e used for a single GeminiDB database. The default memory size is 512 MB.
state.backend.gemini.heap.size.mb=512
# Configure the JVM parameters. Recommended configurations:
blink.job.option=-yD env.java.opts.taskmanager='-XX:NewRatio=3 -XX:SurvivorRatio=3 -X
X:ParallelGCThreads=8 -XX:+UnlockDiagnosticVMOptions -
XX:ParGCCardsPerStrideChunk=4096 -XX:+UseCMSInitiatingOccupancyOnly -
XX:CMSInitiatingOccupancyFraction=75 -Djdk.nio.maxCachedBufferSize=10240'
```

Major bug fixes

- Fixes the bug that causes the Calc operator to encounter a null pointer exception (NPE) during code generation.
- Fixes the bug that requires complete rows to be used for state storage.
- Fixes a bug in the `code splitting` component. When JavaCodeSplitter converts `local variables` to `member fields`, **JavaCodeSplitter** does not process the `local variables`

inside a `"for each" control`. This bug causes invalid calculations when you use `DISTINCT` filtering.

1.11. Blink 3.3.0

This topic lists the major features and changes in Blink 3.3.0 for Realtime Compute.

Major features

- **Slim CU mode**
  - You can set the expected number of CUs for the initial execution plan. The specified number of CUs determines the initial parallelism settings of a job when the execution plan is generated.
  - You can use the resource allocation for the slim CU mode. If only one parallel instance is set for each vertex and the number of required CUs exceeds the specified number of expected CUs, the slim CU mode is enabled. In the slim CU mode, the parallel instances of multiple vertexes automatically run on one slot. All the vertexes are scheduled to run on one TaskManager to reduce the consumed resources.

Feature changes

- **Automatic scaling**
  - The maximum CUs for the automatic scaling feature are changed from the maximum CUs of a job to the maximum CUs that are specified in the execution plan. This resolves the issue that jobs fail to be started because of the limit for CUs. After this change is made, the CUs consumed by a job may exceed the maximum CUs of the automatic scaling feature.
  - Optional job parameters are added for the automatic scaling feature. You can use the added parameters to manually disable the scaling down feature. This ensures the stable running of jobs. The added parameters are `healthmanager.resource.scale.down.enabled` (controls resource scaling down) and `healthmanager.parallelism.scale.down.enabled` (controls parallelism scaling down).
  - Resources can be manually configured and the automatic scaling feature can be enabled for the jobs that are created based on the DataStream API. As of Blink 3.3.0, you can edit the resource plans and enable the automatic scaling feature for the jobs that are created based on the DataStream API. The automatic scaling feature for the jobs that are created based on the DataStream API is only available for trial use.

- **Read first rows of duplicate records based on the Rowtime field**

  You can read the first rows of duplicate records for deduplication based on the Rowtime field. The Rowtime field indicates the event time and you can perform window operations only on the data that has the event time attribute. After node deduplication, you can still perform window operations on the records.

- **Case insensitive for SQL code**
  As of Blink 3.3.0, SQL code is case insensitive. Compilation errors may occur due to this change. If you use uppercase and lowercase letters to distinguish variables or identifiers, compilation errors occur.
  - Example

```
    t.taobao_bind AS taobao_bind,
    t.et_taobao_bind AS et_taobao_bind
FROM
    view_t_partner_map_taobao_bind t,
    lateral table (STRING_SPLIT(t.utdids, '\004')) AS T(utdid0);

INSERT INTO
```

- Error message

```
org.apache.flink.table.api.ValidationException:
************
ERR_ID:
     SQL-00120001
CAUSE:
     SQL validation failed:
     From line 3, column 9 to line 3, column 69: Duplicate relation name 'T' in FROM clause
ACTION:
     Please see descriptions above. If it doesn't help, please contact customer support for this.
DETAIL:
```

1.12. Blink 3.2.3

This topic lists the improvements and bug fixes in Blink 3.2.3 for Realtime Compute. Blink 3.2.3 is released to enhance your development experience.

Improvements

- Resolves the following issue: In the Realtime Compute console, the configuration details in the **Vertex Topology** section of the **Administration** page are inconsistent with those on the **Configuration** tab of the **Development** page.
- Resolves the following issue of Blink 3.2.1: The task ID fails to be displayed on the **Curve Charts** page.
- Resolves the issue: Garbled text exists in the logs on the TaskManager tab of the **Administration** page.
- Resolves the following issue of Blink 3.2.1: The garbage collection log overwrites the debugging result. Before a job file is published, data debugging is performed.
- Resolves the following issue: The automatic scaling feature for exclusive clusters can be enabled only by adding job parameters.
- Resolves the following issue: If LEFT JOIN statements are executed, INNER JOIN statements are displayed in the **Vertex Topology** section on the **Administration** page.
- Resolves the following issue: Users cannot locate the specific row where errors occur in the code editor.

Bug fixes

- Fixes the following bug: The REGEXP_EXTRACT function does not return null if any argument is null or regular expression is invalid.
- Fixes the following bug: Backward slashes (\) of regular expressions in the code are compiled as semicolons (;).
- Fixes the following bug: The debugging result that is printed to the taskmanager.out file is inaccurate. Before a job file is published, data debugging is performed.
- Fixes the following bug: No errors are reported when the LEFT JOIN operation is performed on a dimension table that is not declared as the dimension table.
- Fixes the following bug: Errors occur when the data of the time data type is written to the AnalyticDB for MySQL result table.
- Fixes the following bug in Blink 3.2.1: If a JOIN operation is performed on a MaxCompute dimension table and the ON clause includes fields of the timestamp data type, job running errors occur.
- Fixes the following bug in Blink 3.2.1 and 3.2.2: Job running errors occur when the `minibatch` parameter is used.
- Fixes the following bug: The complex event processing (CEP) syntax passed the check on the Development page, but job running errors occur.
- Fixes the following bug: The `maxRetryTimes` parameter in the ApsaraDB for Hbase result table fails to take effect.

- Fixes the following bug: The group ID is not displayed in Alibaba Cloud Message Queue for Apache Kafka when Blink reads data from Alibaba Cloud Message Queue for Apache Kafka source tables.
- Fixes the following bug: Running errors occur if only one record of the varbinary data type is written to the message queue result table.

1.13. Blink 3.2.1

1.13.1. Blink 3.2.1 release notes

This topic describes the major features of Blink 3.2.1 and its compatibility with the DataStream API and SQL.

Major features

Blink 3.2.1 is the first official version based on the Blink open source code. Blink 3.2.1 offers the following major features:

- Job AutoScale

  Blink 3.2.1 implements the automatic optimization feature by using automatic configuration and automatic scaling. The automatic optimization feature dynamically adjusts the parallelism settings and resources of each operator based on the job running status and amount of input data. This allows you to shorten job delays. In Blink 3.2.1, this feature is in public review.

- Support for the DataStream API
  - Blink 3.2.1 supports the DataStream API. Blink 3.2.1 is developed based on the open source Flink 1.5.1 branch. For information about the DataStream API compatibility between Blink 3.2.1 and Flink 1.5.1, see API compatibility report: Blink 3.2 and Flink 1.5.1.
  - Connectors newly supported by the DataStream API

Connector	Source	Sink
Kafka	Compatible	Compatible
HBase	Not compatible	
JDBC		
RDS<MySQL>		
ES		
MongoDB		

- Connectors newly supported for SQL jobs
  Besides the connectors supported in Blink 2.0, the following connectors are newly supported in Blink 3.2.1.

Connector	Type
ES	DIM
MongoDB	SINK
Redis	DIM

Redis	SINK
Message Queue for Apache RocketMQ V4.2.0	SOURCE
SQL Server	SINK

Compatibility

- For information about DataStream API compatibility, see API compatibility report: Blink 3.2 and Flink 1.5.1.
- For information about SQL compatibility, see Incompatible SQL items: Blink 3.0 and Blink 2.0.

1.13.2. API compatibility report: Blink 3.2 and Flink 1.5.1

This topic describes the test result of API compatibility between Blink 3.2 and Flink 1.5.1.

Scope

- flink-clients
- flink-core
- flink-java
- flink-java8
- flink-optimizer
- flink-scala
- flink-scala-shell
- flink-streaming-java
- flink-streaming-scala
- flink-yarn
- flink-connectors
- flink-filesystems
- flink-formats
- flink-metrics
- flink-queryable-state
- flink-state-backends

Compatibility details

- flink-core
  The total number of tested methods is 6126, among which only one method is incompatible between Blink 3.2 and Flink 1.5.1.

No.	Severity (high, medium, or low)	Old API	Change	Impact

1	Medium	**GenericCsvInputFormat.supportsMultiPaths()**	The supportsMultiPaths() method is removed from the class, and the returned default value changes. The new default value does not support multiple paths.	This change affects the **GenericCsvInputFormat** subclasses that involve multiple paths.

- flink-connector-elasticsearch
  The total number of tested methods is 14, among which only one method is incompatible between Blink 3.2 and Flink 1.5.1.

No.	Severity (high, medium, or low)	Old API	Change	Impact
1	Medium	**ElasticsearchSink**	The class is changed from **ElasticsearchSinkBase<T>** to **ElasticsearchSinkBase<T,org.elasticsearch.client.Client>**.	Flink 1.5.1 does not support the subclasses of the previous class.

flink-json

The total number of tested methods is 34, among which only one method is not compatible between Blink 3.2 and Flink 1.5.1.

No.	Severity (high, medium, or low)	Old API	Change	Impact
1	Medium	**JsonSchemaConverter**	The class is renamed **JsonRowSchemaConverter**.	Flink 1.5.1 does not support the subclasses of the previous class.

flink-streaming-java

The total number of tested methods is 3031, among which four methods are incompatible between Blink 3.2 and Flink 1.5.1.

No.	Severity (high, medium, or low)	Old API	Change	Impact

1	Medium	**TwoInputStreamOperator.processElement1** or **TwoInputStreamOperator.processElement2**	The void return type is changed to the **TwoInputSelection** return type. The endInput1 and endInput2 abstract methods are added to the **TwoInputStreamOperator** interface.	Flink 1.5.1 support **TwoInputStreamOperator** implementers that are not supported by Blink 3.2.
2	Medium	**OneInputStreamOperator** class	The **endInput() abstract** method is added.	Flink 1.5.1 support **OneInputStreamOperator** implementers that are not supported by Blink 3.2.
3	Medium	**StreamOperator** class	The **requireState abstract** method is added.	Flink 1.5.1 support **StreamOperator** implementers that are not supported by Blink 3.2.
4	Medium	**OneInputStreamOperator** class	The **endInput() abstract** method is added.	Flink 1.5.1 support **OneInputStreamOperator** implementers that are not supported by Blink 3.2.

1.13.3. Incompatible SQL items: Blink 3.0 and Blink 2.0

SQL syntax changes

- Syntax changes

  [Over Agg] The window rank function without order by

- Behavior changes
  - [Division to double] | As of Blink 2.2, the division data type can be implicitly converted to the double data type.
  - [Decimal] DDL decimal type default precision changed to (10, 0) | As of Blink 2.2, the default precision for the decimal data type changes.
  - [CEP] A pattern cannot end with a portion that is used for greedy matching. For example, the (a b+) pattern is not supported. To resolve this issue, you can change the pattern to (a b+ c) and define c as not b.
  - [CEP] The WITHIN clause does not support dynamic windows.

Interface changes

- Code refactoring and semantic changes

  [StreamTableSink#emitDataStream returns values changes from void to StreamTableSink]

- Class relocation
  - [Parser inherited com.alibaba.blink.streaming.connectors.common.source.SourceCollector]
  - [Class not found] com/alibaba/blink/exceptions/NotEnoughParamsException
  - [Class not found] com/alibaba/blink/exceptions/UnsupportedTableException
  - [Class not found] org/apache/flink/table/sources/BatchExecTableSource
  - [Class not found] org/apache/flink/table/functions/aggfunctions/DoubleSumWithRetractAggFunction
  - [Class not found] org/apache/flink/table/functions/Monotonicity
  - [Class not found] Lcom/alibaba/blink/cache/Cache
  - [Class not found] org/apache/flink/table/row/GenericRow
- Implementation changes
  - [Method not found] com.alibaba.blink.table.api.RichTableSchema.getColumnTypes
  - [Method not found] Lorg/apache/flink/table/types/DataType.of
  - [Verification] java.lang.VerifyError: class com.koubei.blink.connector.sls.CustomTableFactory overrides final method setClassLoader
  - [Class not found] com/aliyun/odps/OdpsException

Connectors

- [ODPS] ODPSTableSink stream mode do not support overwrite
- [ODPS] Only batch mode support overwrite

2.Product Introduction

2.1. Overview

This topic describes the benefits and network architecture of the exclusive mode.

Shared mode

In shared mode, users share physical resources such as networks, disks, CPUs, and memory in a computing cluster. Account management and control groups (cgroups) are used to implement resource isolation and security management. For account, business, and data security concerns, the shared mode does not support user-defined functions (UDFs).

> **Note**
>
> As of December 24, 2019, Realtime Compute for Apache Flink in shared mode is no longer available. You cannot purchase projects in this mode. You can only scale out, scale in, or renew existing shared-mode projects. We recommend that you purchase the exclusive mode or semi-managed Flink mode of Realtime Compute for Apache Flink based on your business requirements.

Exclusive mode

- Benefits

  In exclusive mode, an independent computing cluster is created on an Alibaba Cloud Elastic Compute Service (ECS) instance. A single user can exclusively use physical resources such as networks, disks, CPUs, and memory in the computing cluster. The resources of the user are isolated from those of other users. the exclusive mode provides the following benefits:

  - Adaption to various hardware

    Leverages the capabilities of Alibaba Cloud in hardware-specific optimization such as CPU-to-memory ratio and GPU or FPGA. This solves hardware adaptation issues.

  - Isolation between users

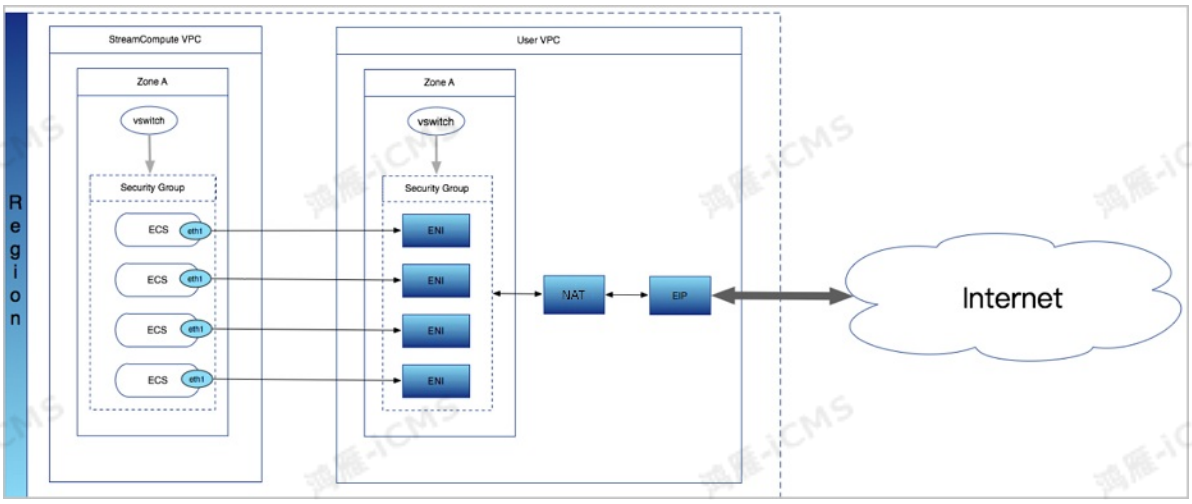
    Allows you to use a VPC and exclusive computing resources. In addition, you can connect your development platform to the VPC to meet your business requirements.

  - Support for UDFs

    Isolates your network and physical machines from those of other users. This way, you can use UDFs and underlying APIs to meet your business requirements. For more information about UDFs, see Overview.

  - Rich features
    - Extract, transform, load (ETL) in a data lake: You can use **Flink SQL and UDFs** to develop ETL tasks.
    - Computing of heterogeneous data sources: Data can be read from heterogeneous data sources for analysis. For example, Realtime Compute for Apache Flink can read archived logs from Object Storage Service (OSS) buckets and associate the logs with high-risk IP addresses in an ApsaraDB for HBase database to analyze web attacks.
    - Multiple upstream and downstream data stores are supported, such as Create a Message Queue for Apache Kafka source table and Create a Message Queue for Apache Kafka result table.

- Architecture of Realtime Compute for Apache Flink in exclusive mode



  - For Realtime Compute for Apache Flink in exclusive mode, all your purchased ECS instances are hosted in the VPC of your Realtime Compute for Apache Flink cluster. In this mode, you cannot log on to these ECS instances.
  - When you create a cluster, you can apply for an elastic network interface (ENI) within your account to access all resources in the VPC to which the ENI belongs.
  - To access the Internet, you can bind a NAT gateway and an elastic IP address (EIP) to the ENI. For more information, see Associate an EIP with an Internet NAT gateway.

    > **Note**
    >
    > You are charged for the use of the ENI only when your Realtime Compute for Apache Flink cluster accesses the Internet.

  - To access services of other security groups in the VPC, you must configure inbound and outbound rules for the security group.

Differences between exclusive mode and shared mode

Item	Exclusive mode	Shared mode
Development of custom features	Supports UDFs and APIs. This allows you to flexibly develop jobs.	Not supported

Network type	Supports Alibaba Cloud VPCs. **Note** Users in a Realtime Compute for Apache Flink cluster in exclusive mode can access only the upstream and downstream storage resources in the same region and VPC as the cluster. If you want to access resources in another VPC, you must configure port numbers of security groups and use Express Connect to access the VPC.	Not supported
Machine type selection	Allows you to select a machine type when you purchase Realtime Compute for Apache Flink in exclusive mode.	Not supported
Supported data sources	• DataHub • Log Service • Message Queue (MQ) • Message Queue for Apache Kafka **Note** For more information about source tables of Realtime Compute for Apache Flink, see Overview of source tables.	• DataHub • Log Service • MQ **Note** For more information about source tables of Realtime Compute for Apache Flink, see Overview of source tables.

Supported data outputs	• AnalyticDB for MySQL • DataHub • Log Service • MQ • Tablestore • ApsaraDB RDS or DRDS • TSDB • HybridDB for MySQL • Kafka • ApsaraDB for HBase • Elasticsearch **Note** For more information about result tables of Realtime Compute for Apache Flink, see Overview of result tables.	• AnalyticDB for MySQL • DataHub • Log Service • MQ • Tablestore • ApsaraDB RDS or DRDS • TSDB • HybridDB for MySQL **Note** For more information about result tables of Realtime Compute for Apache Flink, see Overview of result tables.
Supported regions	• Pay-as-you-go: China (Hangzhou), China (Beijing), China (Shanghai), and China (Shenzhen) • Subscription: China (Hangzhou), China (Beijing), China (Shanghai), China (Shenzhen), China (Zhangjiakou), China (Hong Kong), and Singapore (Singapore).	China (Shenzhen) **Note** As of December 24, 2019, Realtime Compute for Apache Flink in shared mode is no longer available. You cannot purchase projects in this mode. You can only scale out, scale in, or renew existing shared-mode projects. We recommend that you purchase the exclusive mode or semi-managed Flink mode of Realtime Compute for Apache Flink based on your business requirements.
Specification	Provides an independent real-time computing engine and allows each user to perform stream processing in an independent ECS cluster.	Provides a large cluster to allow users to share public resources.
Isolation	Provides strong isolation. Resources in an ECS instance such as networks and security groups are isolated from the resources of other users.	Provides weak isolation. Network resources cannot be isolated.
Customer groups	Big data teams that have advanced development technologies and demand for flexible and controllable development.	Teams or individuals that only want to realize streaming business and have no special requirements for the development process.

Cost	Has a higher cost than the shared mode. In high availability mode, each user must bear the cost of three additional master nodes.	Users only need to pay for the computing service.
Cluster management and maintenance	Requires cluster management and network configuration skills.	Cluster management is not required.

2.2. Development history

This topic describes the development history of Realtime Compute.

Alibaba Cloud Realtime Compute offers an end-to-end solution of stream processing based on Flink, from job development to administration. Based on many years of experience of Alibaba Group in big data technologies and business scenarios, Realtime Compute allows you to take advantage of the powerful capabilities of advanced computing engines. By leveraging the experience and expertise of Alibaba Group in streaming data services, you can easily and quickly utilize the benefits of stream processing to accelerate the growth of big data services.

- Beginning: Real-time big screen service of the Double 11

  Realtime Compute has its beginnings in the big screen service of the Double 11. With years of experience and development, the small team that once provided the real-time big screen service and limited real-time reporting services has become an independent and reliable cloud computing team. Realtime Compute provides an end-to-end cloud solution of stream processing based on years of experience in real-time computing products, architecture, and business scenarios. We strive to offer powerful support for small and medium-sized enterprises (SMEs) in terms of real-time big data processing.

- Early stage: Development based on open source Flink

  Alibaba Group adopted open source Flink to support the big screen service during the Double 11. Flink code was created for stream processing. During the early stages, stream processing services were provided on a small scale. Developers used Flink APIs to create stream processing jobs. Therefore, developers must have proficient technical skills, handle debugging challenges, and perform large amounts of repetitive tasks.

- Continuous optimization: Development based on Flink APIs

  To handle large amounts of repetitive work, Alibaba Group engineers started working on data encapsulation and abstraction. Based on Flink APIs, they developed a large number of reusable components for data statistics, such as the basic programming components for simple filtering, aggregation, and windows. Based on these components, an XML description language is provided. With this design, Realtime Compute users can use Extensible Markup Language (XML) to describe and integrate Flink components, and create end-to-end real-time computing processes. This programming method eliminates large amounts of repetitive development work that is required at the underlying layer, and reduces the requirements for development skills. This programming method is different from the SQL method that is most familiar to data analysts. Therefore, analysts must learn more about the programming components and XML syntax.

- Maturity: Flink SQL development

Any emerging technology is only adopted by a small group in the beginning. With the growth of this technology and the reduction in adoption costs, it will be widely accepted. Therefore, Alibaba Cloud engineers are working to enable stream processing technologies to be widely adopted by improving the technology and decreasing adoption costs. Thanks to years of experience in relational databases, Alibaba Group engineers developed Flink SQL to replace the programming method that is based on XML and Flink components. Flink SQL allows you to write SQL code for real-time computing and data processing. All these improvements are integrated into Flink, the core computing engine of Realtime Compute. For this computing engine, a single cluster includes up to thousands of machines. An average of hundreds of billions of messages can be processed per day, and the amount of data that is processed per day nearly reaches the PB level. Flink clusters have become the core stream processing clusters of Alibaba Group.

Flink SQL offers the following benefits:

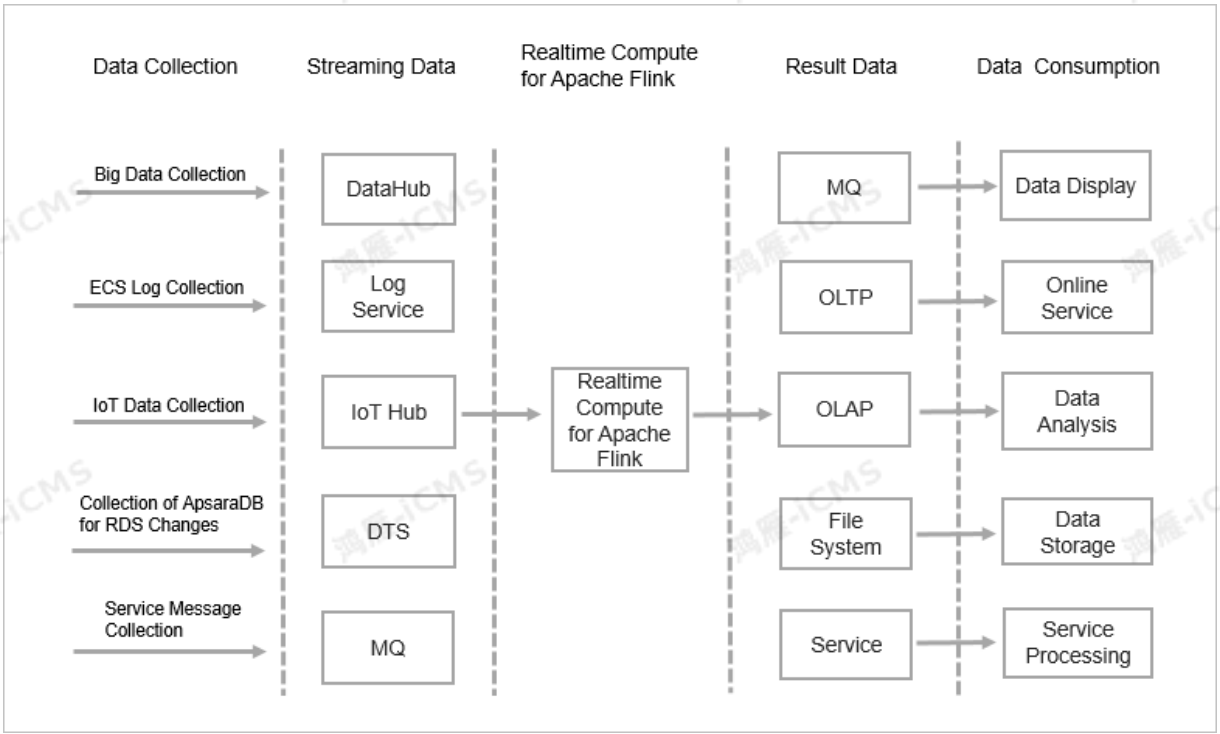
- Flink SQL supports a wide range of SQL functions, which improves the technical maturity of users.
- You can use familiar SQL models for easy adoption of Realtime Compute.

2.3. Workflow

This topic describes the architecture and data links in the workflow of Alibaba Cloud Realtime Compute for Apache Flink.

Architecture

The following figure shows the architecture of Realtime Compute for Apache Flink.



1. **Data collection**

You can use streaming data collection tools to collect and send streaming data in real time to a publish-subscribe system for big data analysis. This publish-subscribe system continuously produces events for Realtime Compute for Apache Flink in the downstream to trigger stream processing jobs. The Alibaba Cloud big data ecosystem offers publish-subscribe systems for big data analysis in different scenarios. Realtime Compute for Apache Flink integrates multiple publish-subscribe systems shown in the preceding figure and therefore can integrate various data streams.

> **Note**
>
> For example, you can directly connect Realtime Compute for Apache Flink to LogHub of Log Service to quickly integrate and use ECS logs.

2. **Stream processing**

   Data streams continuously enter Realtime Compute for Apache Flink for real-time processing. At least one data stream must enter Realtime Compute for Apache Flink to trigger a Realtime Compute for Apache Flink job. In complex business scenarios, Realtime Compute for Apache Flink allows you to perform JOIN operations on the static data of data stores.

   > **Note**
   >
   > For example, you can perform JOIN operations on DataHub and ApsaraDB RDS data based on the primary key of streaming data.

3. **Real-time integration**

   Realtime Compute for Apache Flink can directly write the result data of stream processing into the destination data store. Realtime Compute for Apache Flink integrates Alibaba Cloud ecosystems such as OLTP (for example, ApsaraDB RDS), NoSQL (for example, Tablestore), OLAP (for example, AnalyticDB for MySQL), Message Queue (for example, DataHub and ONS), and MassiveStorage (for example, OSS and MaxCompute). This minimizes the end-to-end data latency and complexity of data links and ensures real-time data processing.

4. **Data consumption**

   After the result data of stream processing is written into a storage system, you can use customized applications to manage the result data. You can use a storage system to access the result data, data transmission system to receive the result data, or alerting system to send alerts.

Data links

Some Alibaba Cloud ecosystems do not support Realtime Compute for Apache Flink. You must convert the streaming data of these storage systems to other data types.

- LogService

  Log Service is an end-to-end service for log data. It allows you to quickly collect, transfer, query, consume, and analyze log data. For more information about how to use logs to collect streaming data, see Data collection overview.

- IoTHub

  IoT Hub helps developers build secure data channels to implement bidirectional communications between the cloud and terminal devices. The terminal devices include sensors, actuators, embedded devices, and smart home appliances. You can use the IoT Hub rules engine to easily transfer IoT data to DataHub and use Realtime Compute for Apache Flink and MaxCompute to perform computations on the data. For more information about how to transfer IoT data to DataHub, see Configure a data forwarding rule.

- MQ

  Alibaba Cloud Message Queue is a complete messaging service. It provides features such as publishing and subscription, message tracing, resource statistics, timing (latency), and monitoring and alerting based on distributed clusters in high availability mode.

2.4. Upstream and downstream data stores

Realtime Compute for Apache Flink supports a wide range of upstream and downstream data stores.

- Source tables
  - Create an Oracle database source table
  - Create a Log Service source table
  - Create a Hologres source table
  - Create a source table
  - Create a Message Queue for Apache Kafka source table
  - Create a Tablestore source table
  - Create a full MaxCompute source table
  - Create an incremental MaxCompute source table
- Result tables
  - Create an AnalyticDB for MySQL V2.0 result table
  - Create a Hologres result table
  - Create an Oracle database result table
  - Create a Log Service result table
  - Create a ApsaraMQ for RocketMQ result table
  - Create a Tablestore result table
  - Create an ApsaraDB RDS result table
  - Create a MaxCompute result table
  - Create an ApsaraDB for HBase result table
  - Create an Elasticsearch result table
  - Create a TSDB result table
  - Create a Message Queue for Apache Kafka result table
  - Create a HybridDB for MySQL result table
  - Create an ApsaraDB RDS for SQL Server result table
  - Create an ApsaraDB for Redis result table
  - Create an ApsaraDB for MongoDB result table
  - Create an AnalyticDB for MySQL V3.0 result table
  - Create an AnalyticDB for PostgreSQL result table
  - Create a custom result table
  - Create an InfluxDB result table
  - Create a Phoenix5 result table
- Dimension tables
  - Create a Hologres dimension table
  - Create a Tablestore dimension table

- Create an ApsaraDB RDS for MySQL dimension table
- Create an ApsaraDB for HBase dimension table
- Create a MaxCompute dimension table
- Create an ApsaraDB for Redis dimension table
- Create a Phoenix5 dimension table
- Create an AnalyticDB for MySQL V3.0 dimension table
- Create an Elasticsearch dimension table

2.5. Product security

Realtime Compute for Apache Flink ensures account, business, and data security for end-to-end real-time computing.

Account security involves Realtime Compute for Apache Flink and data stores.

- Account security of Realtime Compute for Apache Flink

  Only Alibaba Cloud accounts can be used as Realtime Compute for Apache Flink accounts. The account information includes a username and its password or a username and a signature key. HTTPS is used to secure account information. For more information about account security of Realtime Compute for Apache Flink, see Grant permissions to a RAM user.

- Account security of data stores

  In Realtime Compute for Apache Flink, the accounts of data stores are used to create source and result tables. Realtime Compute for Apache Flink provides Resource Access Management (RAM) and Security Token Service (STS) to prevent the leakage of your business data due to the loss of account information. For more information about account security for data stores, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

Business security

Business security of Realtime Compute for Apache Flink is used to isolate projects and secure business processes.

- Project isolation

  Realtime Compute for Apache Flink projects are isolated based on project permissions. Only users that belong to a project can access or manage authorized sub-product entities in the project. Project-level resource isolation ensures that other users do not interfere with your operations.

  > **Note** For example, if the data amount increases dramatically when a job of a user is running, the CPU utilization of the job is increased. Due to resource isolation, the CPU utilization of jobs of other users is not affected.

- Business process

  Realtime Compute for Apache Flink provides separate pages for data development and administration to clearly demonstrate the entire development process of stream processing in its console. This guarantees a complete and secure business process.

  - Code version

    Realtime Compute for Apache Flink allows you to compare code versions and roll back to an earlier version. This helps you trace the code and rectify faults.

- Standalone debugging tool in the IDE

  Realtime Compute for Apache Flink offers a debugging tool in the integrated development environment (IDE), which allows you to debug the code without affecting online data. You can specify data for source tables, dimension tables, and result tables to create a job, and then debug the data offline. This ensures that running jobs are not affected.

- Job publishing process

  The job publishing process is secure. You can debug the code without affecting online data. After you debug the new code, you can publish the job and view it on the Administration page of the Realtime Compute for Apache Flink development platform. Realtime Compute for Apache Flink jobs that are running do not directly use the new code. To use the new code, you must stop the jobs and then restart them with the new code. .

Data security

Realtime Compute for Apache Flink ensures the security of its system data and business data.

- System data security

  Realtime Compute for Apache Flink ensures its data security in the following aspects:

  - HTTPS is used to secure transmission links.
  - The Advanced Encryption Standard (AES) is used to encrypt information about the connections with data stores. This helps prevent the leakage of sensitive information.
  - Realtime Compute for Apache Flink has passed comprehensive and in-depth attack tests.
  - Alibaba Cloud security team provides security services for Realtime Compute for Apache Flink.

- Business data

  Realtime Compute for Apache Flink does not store the business data of users. The security of business data is ensured by Alibaba Cloud storage systems. For more information, see the security models and best security practices of Alibaba Cloud storage systems.

2.6. Limits

This topic describes the service scope and limits of Realtime Compute for Apache Flink, including the limits on CU processing capabilities and job creation.

Evaluate the impact of the limits on your business carefully.

- If you want to use UDXs, you must purchase Realtime Compute for Apache Flink in exclusive mode. For more information about UDXs, see Overview.
- Realtime Compute development platform supports only the Google Chrome browser.

Supported regions

The regions supported by Realtime Compute for Apache Flink vary based on your business requirements.

- Pay-as-you-go for the exclusive mode: China (Hangzhou), China (Beijing), China (Shanghai), and China (Shenzhen).
- Subscription for the exclusive mode: China (Hangzhou), China (Beijing), China (Shanghai), China (Shenzhen), China (Zhangjiakou-Beijing Winter Olympics), China (Hong Kong), and Singapore (Singapore).

CU processing capabilities

Test results of Realtime Compute for Apache Flink demonstrate that the CU processing capabilities vary based on the complexity of businesses.

- For simple operations such as single-stream filtering and string conversion, one CU can process 10,000 data records per second.
- For complex operations such as operations that use a JOIN clause, GROUP BY clause, or window function, one CU can process 1,000 to 5,000 data records per second.

Limits on job and task quantities

Realtime Compute for Apache Flink has the following limits on jobs, task versions, and task pages opened in an integrated development environment (IDE) under a project:

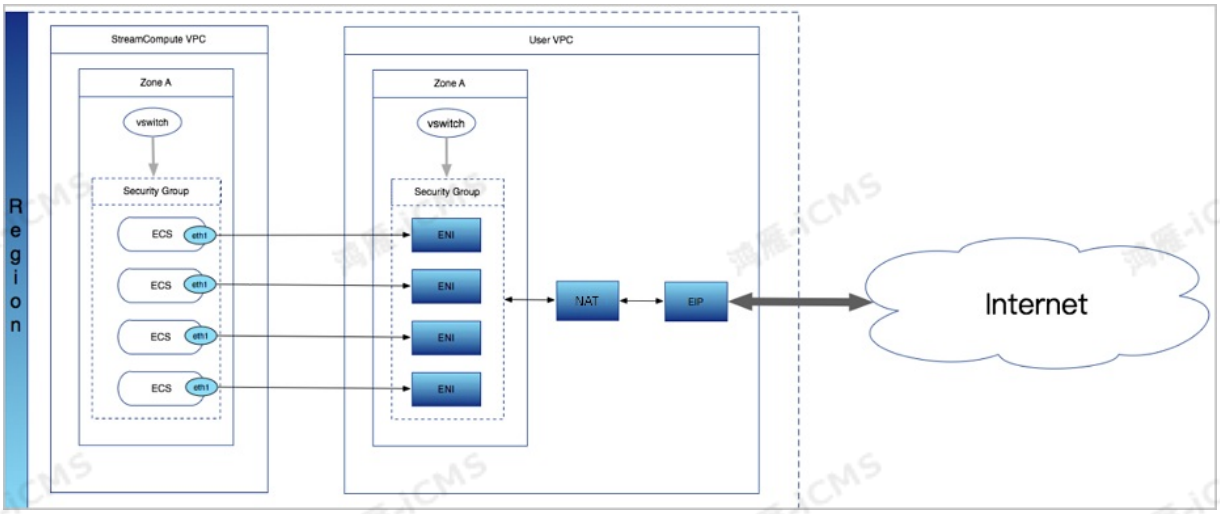
- A maximum of 100 jobs can be created under a project.
- A maximum of 50 folders are allowed under a project. The number of folder hierarchies cannot exceed 5.
- A maximum of 50 UDXs or JAR packages are allowed under a project.
- A maximum of 50 data stores can be registered under a project.
- A maximum of 20 historical versions can be saved in a job.

2.7. Architecture of Realtime Compute for Apache Flink in exclusive mode (phased-out)

This topic describes the architecture of Realtime Compute for Apache Flink in exclusive mode.

Architecture

The following figure shows the architecture of Realtime Compute for Apache Flink in exclusive mode.



- If you use Realtime Compute for Apache Flink in exclusive mode, all your purchased Elastic Compute Service (ECS) instances are fully hosted in the virtual private cloud (VPC) in which your Realtime Compute for Apache Flink cluster resides. In this mode, you cannot log on to your purchased ECS instances.

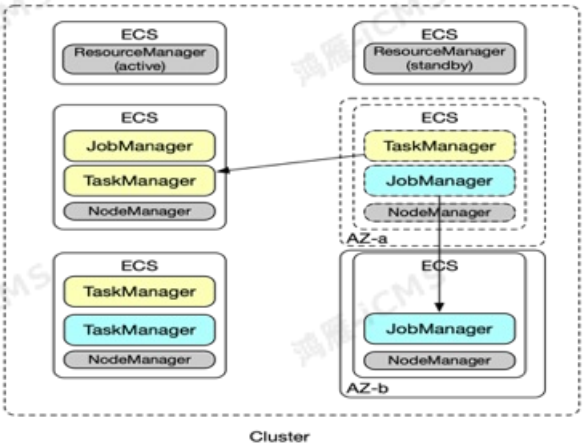
- When you create a Realtime Compute for Apache Flink cluster, Realtime Compute for Apache Flink applies for an elastic network interface (ENI) under your account. You can use this ENI to access all the resources in your VPC.
- To allow your Realtime Compute for Apache Flink cluster to access the Internet, you can bind a network address translation (NAT) gateway and an elastic IP address (EIP) to the ENI. For more information, see Associate an EIP with a NAT gateway.
- The ENI belongs to an independent security group under your account. To access the services of other security groups in the VPC, you must configure inbound and outbound rules for the security group.

> **Note**
>
> You are charged for the use of the ENI only when your Realtime Compute for Apache Flink cluster accesses the Internet.

Benefits

- End-to-end real-time data computing and development
  - Provides real-time data processing capabilities based on Flink SQL, which implements automatic data recovery. This ensures accurate data processing even if failures occur.
  - Supports multiple built-in functions, such as string, date, and aggregate functions.
  - Supports various window types, such as tumbling, sliding, and session windows.
  - Provides accurate control over computing resources, which ensures resource isolation for jobs.
  - Provides the following key performance metrics that are superior to the metrics of Apache Flink:
    - The data computing latency can be indicated in subseconds.
    - The throughput of a single job can reach millions of records per second. A single cluster can consist of thousands of servers.
  - Deeply integrates various cloud data storage systems such as DataHub, Log Service, ApsaraDB RDS, Tablestore, and AnalyticDB for MySQL. This allows you to read and write data from and to these systems in a convenient manner.
- Fully-managed real-time computing service
  - Uses a fully-managed stream computing engine.
  - Allows you to run and query streaming data without the need to provision or manage infrastructures.
  - Allows you to activate streaming data processing services with one click.
  - Integrates features such as data storage, data development, data O&M, and monitoring and alerting. This reduces both the trial and migration costs of stream processing.
  - Isolates and protects the managed and running services of different tenants.
- Reduced manpower and cluster costs
  - Significantly optimizes the SQL execution engine to provide computing jobs that are more cost-effective than native Flink jobs.
  - Significantly reduces development and operation costs, which are much lower than the costs of open source streaming frameworks.
- High availability

If an ECS instance is abnormal or a Realtime Compute for Apache Flink job is recovered from a failure or is resumed, you can use the JobManager or a TaskManager on an available ECS instance in the same zone to ensure high availability for jobs. You can also use the JobManager or a TaskManager on an available ECS instance in a different zone or region to ensure high availability across zones.



3.Pricing

3.1. Billing unit

This topic describes the billing unit of Realtime Compute for Apache Flink.

The basic billing unit of Realtime Compute for Apache Flink is compute unit (CU), which indicates computing resources. One CU is equal to **1 CPU core and 4 GB of memory**. The number of CUs determines the computing capability of the underlying system of Realtime Compute for Apache Flink.

The number of CUs that is consumed by a Realtime Compute for Apache Flink job varies based on the queries per second (QPS) of the input data stream of the job, the computing complexity, and the distribution of the input data. To estimate the processing capability of one CU in Realtime Compute for Apache Flink, select one of the following methods based on the operation complexity:

- For simple operations such as single-stream filtering and string conversion, one CU can process 10,000 data records per second.
- For complex operations such as operations that use a JOIN clause, GROUP BY clause, or window function, one CU can process 1,000 to 5,000 data records per second.

You can estimate the number of CUs that you need to purchase based on your business scale and the preceding computing capability.

> **Note**
> - The preceding computing capability estimate refers only to the internal processing capability of Realtime Compute for Apache Flink. The external data read and write capabilities are not included. The external data read and write efficiency may affect the estimation of the computing capability of Realtime Compute for Apache Flink.
>   - If you want to use Realtime Compute for Apache Flink to read data from Log Service but the query quota of Log Service is limited, the overall computing capability of Realtime Compute for Apache Flink is subject to the capability allowed by Log Service.
>   - If the number of connections or transactions per second (TPS) is limited for the ApsaraDB RDS database that Realtime Compute for Apache Flink references, the throughput of Realtime Compute for Apache Flink is limited by the throttling of the ApsaraDB RDS database.
> - If you use window functions in a Realtime Compute for Apache Flink job, the number of CUs consumed in the job is greater than that consumed in a simple job. We recommend that you purchase at least four CUs for such a job.

3.2. Billing methods

This topic describes the billing methods of Realtime Compute for Apache Flink.

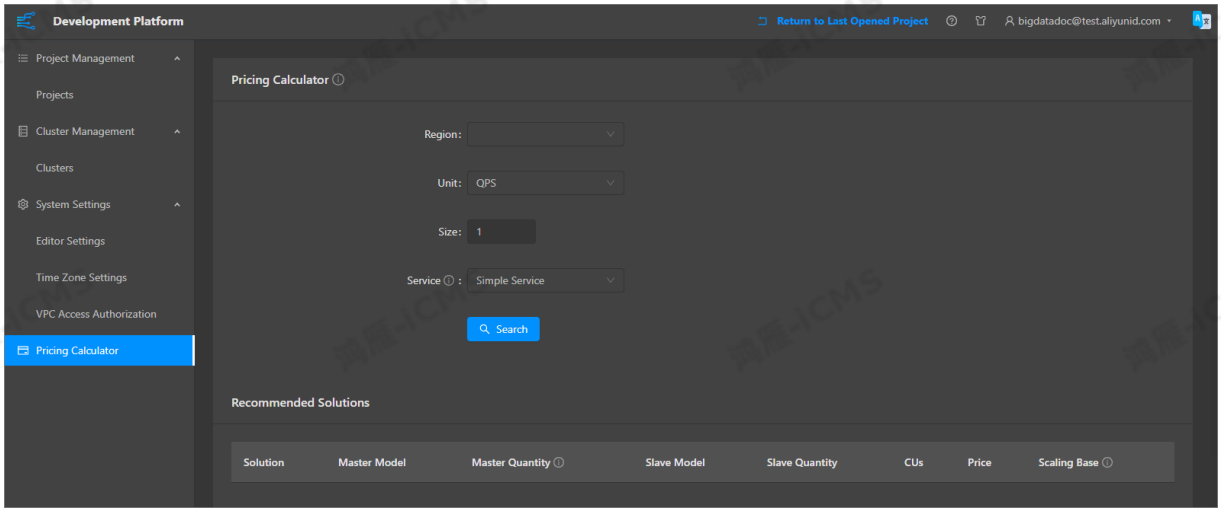
> **Important** The system sends you a notification when you have an overdue payment within your account. We recommend that you promptly pay your overdue bills to prevent your projects from being released. Take note that your projects may be released at a system-selected time after the payment due date.

The following table lists the unit price for each node model.

Region	Node model (master/slave)	Price (USD/month)

Japan (Tokyo)	4 CPU cores, 16 GB of memory	187.16
	8 CPU cores, 32 GB of memory	340.96
	16 CPU cores, 64 GB of memory	648.56
	32 CPU cores, 128 GB of memory	1263.75
	56 CPU cores, 224 GB of memory	2185.18
Malaysia (Kuala Lumpur)	4 CPU cores, 16 GB of memory	164.32
	8 CPU cores, 32 GB of memory	288.07
	16 CPU cores, 64 GB of memory	535.57
	32 CPU cores, 128 GB of memory	1030.60
	56 CPU cores, 224 GB of memory	1773.04
Germany (Frankfurt)	4 CPU cores, 16 GB of memory	187.90
	8 CPU cores, 32 GB of memory	332.44
	16 CPU cores, 64 GB of memory	621.51
	32 CPU cores, 128 GB of memory	1199.64
	56 CPU cores, 224 GB of memory	2066.91

You can also log on to the Realtime Compute development platform console and use the selected cluster configuration in the Recommended Solutions section on the **Pricing Calculator** page. The following figure shows the Pricing Calculator page.



3.3. Specification selection

This topic describes how to select specifications when you configure a Realtime Compute for Apache Flink cluster in exclusive mode. It also provides precautions during the configuration.

Background information

A Realtime Compute for Apache Flink cluster functions as a primary/secondary distributed cluster, which consists of master and slave nodes.

- Master nodes manage cluster resources and interactions among slave nodes, but are not used for computing.
- Slave nodes are used for computing.

> **Note**
>
> A slave node cannot use all of its resources for computing because the operating system and interactions with other nodes also consume resources.

Usage notes

- The slave node specifications determine the scaling configuration. For example, if your slave node specifications are 8 CPU cores and 32 GB memory, you can add or remove only a specific number of nodes of this configuration for each scaling operation. In this case, the number of available compute units (CUs) increases or decreases by the value that is calculated by using the following formula: Number of nodes added or removed × 6 CUs.
- Three master nodes in a Realtime Compute for Apache Flink cluster support failover in case of a failure. This ensures cluster stability. If you configure a Realtime Compute for Apache Flink cluster with three master nodes, Alibaba Cloud provides you with a Service Level Agreement (SLA) guarantee.
- You cannot change the number of master nodes in a Realtime Compute for Apache Flink cluster.

Select specifications

The configuration of a Realtime Compute for Apache Flink cluster in exclusive mode can be calculated in CUs. One CU is equivalent to 1 CPU core and 4 GB memory. You can configure master and slave nodes as required based on the following computing logic. Realtime Compute for Apache Flink provides a pricing calculator to help you configure your cluster to be as cost-effective as possible.

> **Note**
>
> If you are a new user, you can determine the number of CUs based on the queries per second (QPS) and complexity of your business logic, and then configure the cluster. For more information, see Billing methods. Processing capability of one CU in Realtime Compute for Apache Flink:
>
> - For simple operations such as single-stream filtering and string conversion, one CU can process 10,000 data records per second.
> - For complex operations such as operations that use a JOIN clause, GROUP BY clause, or window function, one CU can process 1,000 to 5,000 data records per second.

The following tables list the empirical CU quantities for different master and slave node specifications.

- You must configure at least two slave nodes for a Realtime Compute for Apache Flink cluster in exclusive mode. Therefore, the minimum computing capability of this cluster is 6 CUs, which is calculated by using the following formula: 3 CUs × 2.

Slave node specifications	Available CUs
4 CPU cores, 16 GB memory	3
8 CPU cores, 32 GB memory	6
16 CPU cores, 64 GB memory	13
24 CPU cores, 96 GB memory	21
32 CPU cores,128 GB memory	28
56 CPU cores, 224 GB memory	52
64 CPU cores, 256 GB memory	60

> **Note**
> The empirical data is for reference only.

- The master node specifications are restricted by the maximum number of CUs allowed in a cluster. The following table lists empirical CU quantities for different master node specifications.

Master node specifications	Maximum number of CUs allowed in a cluster
4 CPU cores, 16 GB memory	80
8 CPU cores, 32 GB memory	160
16 CPU cores, 64 GB memory	800
24 CPU cores, 96 GB memory	More than 800

> **Note**
> The empirical data is for reference only.

3.4. Renewal

3.4.1. Manual renewal

This topic describes how to manually renew a Realtime Compute for Apache Flink instance.

Renewal is to extend the duration for a project after it expires. The renewal duration is measured in months or years, and the minimum renewal duration is one month.

> **Important** In the Realtime Compute for Apache Flink console, the **Overview** page shows the remaining days of your projects. The system sends you a notification if you have an overdue payment under your account. We recommend that you pay off your overdue bills to prevent your projects from being released. Note that your projects may be released at a system-selected time after the payment due date.

Renew a Realtime Compute for Apache Flink instance

1. Go to the **Project Management** page.
   i. Log on to the Realtime Compute development platform.
   ii. Move the pointer over the username in the upper-right corner.
   iii. Click **Project Management**.
2. In the left-side navigation pane, choose **Cluster Management > Clusters**.
3. On the **Clusters** page, find your cluster, click **More** in the **Actions** column, and select **Renew**.
4. On the **Renew** page, specify **Renewal Duration**.
5. Read and select **Realtime Compute Exclusive Mode (Subscription) Agreement of Service**.
6. Click **Pay**.
7. On the **Purchase** page, select a payment method.
8. Click **Purchase**.

3.4.2. Auto-renewal

This topic describes how to automatically renew a Realtime Compute instance. You can enable or disable auto-renewal for an instance or change the auto-renewal duration for an instance.

Enable auto-renewal

1. Log on to the Alibaba Cloud Management Console.
2. In the upper-right corner of the page, click Billing and then **Renew**.
3. On the page that appears, click the **Manual** or **Nonrenewal** tab.
4. In the instance list, find the target instance and click **Enable Auto Renewal** in the **Actions** column.
5. In the **Enable Auto Renewal** dialog box, select your expected renewal duration from the **Unified Auto Renewal Cycle** drop-down list.
6. Click **Auto Renew**.

Disable auto-renewal

1. Log on to the Alibaba Cloud Management Console.

2. In the upper-right corner of the page, click Billing and then **Renew**.
3. Click the **Auto** tab.
4. In the instance list, find the target instance and click **Enable Manual Renewal** in the **Actions** column.
5. In the **Enable Manual Renewal** dialog box, click **OK**.

Change the auto-renewal duration

1. Log on to the Alibaba Cloud Management Console.
2. In the upper-right corner of the page, click Billing and then **Renew**.
3. Click the **Auto** tab.
4. In the instance list, find the target instance and click **Edit Auto Renewal** in the **Actions** column.
5. In the **Edit Auto Renewal** dialog box, select your expected renewal duration from the auto-renewal duration drop-down list.
6. Click **OK**.

3.5. Change resource configurations

This topic describes how to scale a Realtime Compute for Apache Flink cluster or project.

Change resource configurations of a cluster

> **Note**
>
> - Fees after the resource configuration change
>   - Scale-out or scale-up (**Master Scale-Up** or **Slave Scale-Out**): You must pay the related upgrade fees.
>   - Scale-in or scale-down (**Master Scale-Down** or **Slave Scale-In**): The refund from the scale-in or scale-down operation operation is returned to your account.
> - After you perform **Master Scale-Up** or **Slave Scale-Out**, you must add required IP addresses to the whitelist of the database. For more information, see Configure a whitelist for accessing storage resources.

You can modify the configuration of **Master Specifications** or **Slave Quantity** to change the configuration of the cluster.

1. Go to the **Project Management** page.
   i. Log on to the Realtime Compute development platform.
   ii. Move the pointer over the username in the upper-right corner.
   iii. Click **Project Management**.
2. In the left-side navigation pane, choose **Cluster Management > Clusters**.
3. On the Clusters page, find your cluster, click **More** in the **Actions** column, and then select **Scale Out** or **Scale In** based on your business requirements.
4. Change resource configurations.

   The following example describes two scale-out or scale-up methods:

- **Master Scale-Up (use higher specifications)**
  a. On the **Update** page, select **Master Scale-Up** for Upgrade Method.
  b. Specify **Master Specifications**.

> **Note**
> - You can change the master specifications but cannot change the number of master nodes.
> - If you use a high availability cluster in which three master nodes are configured, the system upgrades all the three master nodes at the same time when you select **Master Scale-Up**.

- **Slave Scale-Out (increase the number of slave nodes)**
  a. On the **Update** page, select **Slave Scale-Out** for Upgrade Method.
  b. Increase the value of **Slave Quantity**.

> **Note**
> - If the number of CUs is insufficient, we recommend that you select **Slave Scale-Out** and increase the value of **Slave Quantity**.
> - If you are unable to complete payment after you specify Slave Quantity and select Realtime Compute Exclusive Mode (Subscription) Terms of Service, add a vSwitch and enter the ID of the new vSwitch in **vSwitchId**. After verification, you can complete the payment for the scale-out operation. For more information about how to add a VSwitch, see Create and manage a VPC.
> - The slave nodes that you want to add must be located in the same zone as the cluster.
> - You cannot change the slave specifications for an existing Realtime Compute for Apache Flink cluster in exclusive mode. You can only increase the value of **Slave Quantity**. If you want to change the slave specifications, you must purchase a new cluster.

5. Read the terms of service and select Realtime Compute Exclusive Mode (Subscription) Terms of Service.
6. Click **Pay**.

Change resource configurations of a project

> **Note**
> - If the computing capability of existing resources does not meet your business requirements, you can scale out the project to improve the computing capability of the system.
> - If the computing capability of existing computing resources is excessive to meet your business requirements, you can downgrade the resource configuration to reduce costs.

1. Go to the **Project Management** page.
   i. Log on to the Realtime Compute development platform.
   ii. Move the pointer over the username in the upper-right corner.

iii. Click **Project Management**.

2. In the left-side navigation pane, choose **Project Management** > **Projects**.
3. Change resource configurations:
   i. On the Projects page, find your project and click **Scale In/Out** in the **Actions** column.
   ii. In the **Scale In/Out** dialog box, specify Specified CUs.
   iii. Click **OK**.

4.Preparation

4.1. Grant permissions to a RAM user

You can use an Alibaba Cloud account to purchase Realtime Compute for Apache Flink and create projects. You can also use the Alibaba Cloud account to authorize Resource Access Management (RAM) users to access Realtime Compute for Apache Flink projects that are created by the Alibaba Cloud account. This topic describes how to create a RAM user and authorize the RAM user to access Realtime Compute for Apache Flink.

What is a RAM user?

A physical identity that has a fixed ID and credential information. A RAM user represents a person or an application. A RAM user has the following characteristics:

- A RAM user can be created by an Alibaba Cloud account. In this case, the RAM user belongs to the Alibaba Cloud account. A RAM user can also be created by a RAM user or a RAM role that has administrative rights. In this case, the RAM user belongs to the Alibaba Cloud account that creates the RAM user or the RAM role.
- A RAM user does not own resources. Resource usage fees of the RAM user are billed to the Alibaba Cloud account to which the RAM user belongs. A RAM user does not receive individual bills and cannot make payments.
- Before RAM users can log on to the Alibaba Cloud Management Console or call operations, they must be authorized by Alibaba Cloud accounts. After RAM users are authorized, the RAM users can access resources that are owned by the Alibaba Cloud accounts.
- RAM users have independent passwords or AccessKey pairs for logon.
- An Alibaba Cloud account can create multiple RAM users. RAM users can be used to represent employees, systems, and applications within an enterprise.

You can create RAM users and authorize the RAM users to access different resources. If multiple users in your enterprise need to simultaneously access resources, you can use RAM to assign the least permissions to the users. This prevents the users from sharing the username and password or AccessKey pair of an Alibaba Cloud account and reduces the security risks.

Procedure

1. Create a RAM user.

   For more information about how to create a RAM user, see Create a RAM user.

   > **Note**
   > - You must initialize RAM when you use RAM for the first time. For more information, see Configure a password policy for RAM users and Manage security settings of RAM users.
   > - To ensure account security, Realtime Compute for Apache Flink provides the account verification feature. If you do not manage a job for a long period of time, the system sends a text message and an email to you for account verification.

2. Create a custom policy.

   For more information about how to create a custom policy in the RAM console, see Create a custom policy The following code shows a policy of Realtime Compute for Apache Flink:

```
{
   "Version": "1",
   "Statement": [
    {
     "Action": "stream:*",
      "Resource": "acs:stream:*:*:*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:PassRole",
      "Resource": "acs:ram:*:*:*",
      "Effect": "Allow"
   }
  ]
}
```

> **Note**
>
> The policy of Realtime Compute for Apache Flink allows you to grant permissions on different projects to different RAM users. To authorize a RAM user to access a single project, change **Resource** in the preceding code to **"Resource":"acs:stream:*:*:projectname"**. projectname is the name of the project that you want to authorize the RAM user to access.

3. Authorized RAM users or user groups.

   Attach the preceding policy to specified RAM users or RAM user groups. For more information, see Grant permissions to RAM users and Grant permissions to a RAM user group

4. Use the credentials of a RAM user to log on to the Realtime Compute for Apache Flink console.

   In the left-side navigation pane of the RAM console, click **Overview** and view the logon address of the RAM user in the **Account Management** section.

4.2. Activate Realtime Compute for Apache Flink and create a project

This topic describes how to activate Realtime Compute for Apache Flink in exclusive mode. This topic also describes how to create Realtime Compute for Apache Flink clusters and projects in exclusive mode.

Activate Realtime Compute for Apache Flink in exclusive mode

> **Note**
>
> - Realtime Compute for Apache Flink in exclusive mode cannot be purchased from April 28, 2021. You can only scale out, scale in, or renew the existing projects of Realtime Compute for Apache Flink in exclusive mode. If you want to purchase Realtime Compute for Apache Flink, We recommend that you use Flink full hosting for realtime compute.
> - A Realtime Compute for Apache Flink cluster in exclusive mode can access only storage resources in the same virtual private cloud (VPC), region, and security group as the cluster. To allow the cluster to access resources in another VPC, use Express Connect to access the VPC.

After you place an order for Realtime Compute for Apache Flink in exclusive mode, you must create a cluster before you create a project.

1. Activate Realtime Compute for Apache Flink.
   i. Log on to the product page of Realtime Compute for Apache Flink.

   > **Note**
   >
   > Use your Alibaba Cloud account instead of a RAM user to activate Realtime Compute for Apache Flink and create a project.

   ii. Click **Buy Now**.
   iii. Configure the parameters. including the region, master node specifications, number of master nodes, slave node specifications, number of slave nodes, and billing duration based on your business requirements.
   iv. Click **Buy Now**.
   v. Read the terms of service and select **I have read and agree to Realtime Compute Exclusive Mode (Subscription) Agreement of Service**.
   vi. Click **Pay**.
2. Create a Realtime Compute for Apache Flink cluster.
   - Preparations

- After you activate Realtime Compute for Apache Flink in exclusive mode, Realtime Compute for Apache Flink creates a security group in your VPC and applies for an elastic network interface (ENI). For more information, see Overview.

  > **Note**
  >
  > Do not delete the security group or the ENI. Otherwise, the cluster cannot be created.

  - If you have VPCs, specify a VPC for Realtime Compute for Apache Flink.
  - If you do not have a VPC, activate the Alibaba Cloud VPC service. For more information about how to activate the Alibaba Cloud VPC service, see Plan networks.

    > **Note**
    >
    > Make sure that the VPC you created meets the following requirements:
    >
    > - Sufficient Elastic Compute Service (ECS) instances are available in the VPC.
    > - The number of available IP addresses in a vSwitch is greater than or equal to the number of nodes in a Realtime Compute for Apache Flink cluster. For more information, see Configure a whitelist for accessing storage resources and Create and manage a vSwitch.

- You can upload a UDF package to a Realtime Compute for Apache Flink cluster in exclusive mode. To ensure data security, Realtime Compute for Apache Flink stores the UDF package to an Object Storage Service (OSS) bucket. You must specify the OSS bucket. If you do not have OSS buckets, create one first. For more information about how to create an OSS bucket, see Create buckets.
- Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

◦ Procedure

a. After you complete the payment, click **Console**.

b. On the **Clusters** page, click **Create Cluster**.

> **Note**
>
> If no project is created for an order, a red number is displayed on **Create Cluster** in the upper-right corner of the **Clusters** page. The number indicates the number of orders for which no project is created.

c. In the **Select Order** step, select an order in **Order ID** and click Next.

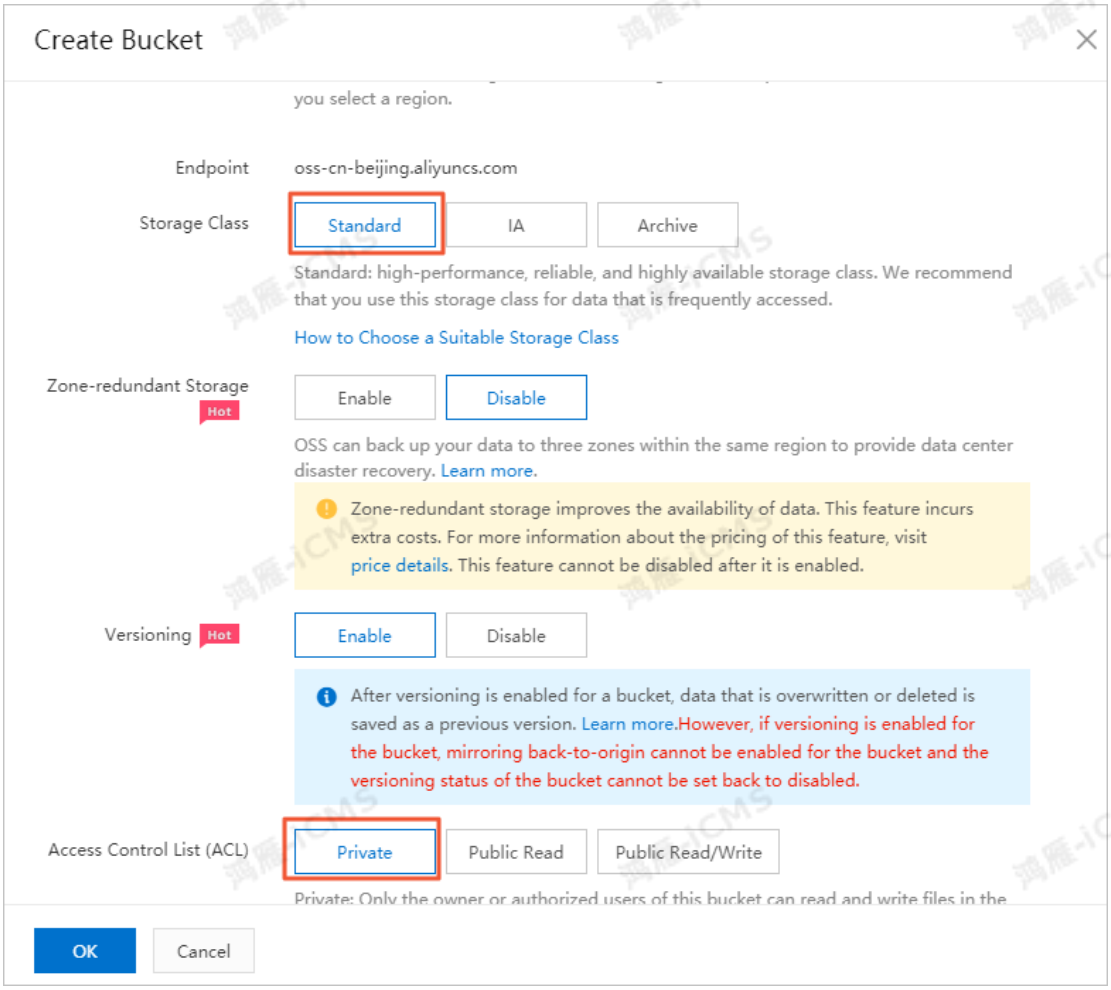
d. In the **Basic Information** step, configure **Cluster Name** and **Cluster Description** and click Next.

e. In the **Cluster Settings** step, enter the configuration information and click Next.

> **Note**
>
> The Realtime Compute for Apache Flink cluster must reside in the same security group, region, and VPC as the upstream and downstream storage that you purchased.

- **OSS Bucket**

  Select an OSS bucket in which you want to store your UDF package. If you do not have OSS buckets, create one first. For more information, see Create buckets. When you create an OSS bucket, you must specify **Standard** for **Storage Class**. We recommend that you specify **Private** for **Access Control List (ACL)**. Do not select **Public Read**.



- **VPC**

  Select the VPC that you want to access and customize the name of the VPC.

> **Note**
>
> Realtime Compute for Apache Flink can identify only custom VPC names.

- **Zone**

  After you properly configure the VPC, the system automatically displays the available zones.

  > **Note**
  >
  > No available zone or vSwitch is displayed in the following scenarios:
  >
  > - ECS instances in the selected zone are insufficient. For more information about how to add an ECS instance, see Create an instance by using the wizard.
  > - The number of available IP addresses in the vSwitch that you select is less than the number of nodes in a Realtime Compute for Apache Flink cluster. For more information, see Configure a whitelist for accessing storage resources and Create and manage a vSwitch.

- **CIDR Block**

  The available CIDR blocks are automatically displayed.

f. In the **Confirm** step, click **Create**.

> **Note**
>
> The cluster is created after it enters the **Running** state from the **Starting** state. This process takes about half an hour.

3. Create a project.
   i. In the left-side navigation pane, choose **Cluster Management > Clusters**. Find the cluster for which you want to create a project and click **Create Project** in the **Actions** column.
   ii. In the **Create Project** dialog box, configure **Project Name** and **Project Description**, and slide the pointer on the right side of **Specified CUs** to specify the required number of compute units (CUs).
   iii. Click **OK**.

4.3. Role authorization

4.3.1. Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode

This topic describes how to assign a Resource Access Management (RAM) role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

Assign a RAM role to an account

You must assign a RAM role to your Alibaba Cloud account before you use Realtime Compute for Apache Flink.

1. Click **Authorize** to go to the authorization page.

> **Note** If you do not assign the default RAM role to your Alibaba Cloud account, the preceding message appears when you use Realtime Compute for Apache Flink.

2. Click **AliyunStreamDefaultRole** and click **Authorize**.

> **Note** After your account is assigned the RAM role, refresh the page in the Realtime Compute for Apache Flink console. Then, you can perform operations in the console.

View the authorization information about the current role

1. Log on to the RAM console.
   - Log on to the RAM console by using your Alibaba Cloud account.
   - Log on to the RAM console as a RAM user.
2. In the left-side navigation pane, click **Roles**. On the Roles page, click **AliyunStreamDefaultRole** in the **Role Name** column of the role list.
3. On the **AliyunStreamDefaultRole** page, click **AliyunStreamRolePolicy** in the Policy column on the **Permissions** tab.
4. On the **Policy Document** tab, view the current policy information of Realtime Compute for Apache Flink.

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ots:List*",
        "ots:DescribeTable",
        "ots:Get*",
        "ots:*Row"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "dhs:Create*",
        "dhs:List*",
        "dhs:Get*",
        "dhs:PutRecords",
        "dhs:DeleteTopic"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:List*",
        "log:Get*",
        "log:Post*"
      ],
      "Resource": "*",
      "Effect": "Allow"
```

```
    },
    {
      "Action": [
        "mns:List*",
        "mns:Get*",
        "mns:Send*",
        "mns:Publish*",
        "mns:Subscribe"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "drds:DescribeDrdsInstance",
        "drds:ModifyDrdsIpWhiteList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "rds:Describe*",
        "rds:ModifySecurityIps*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "vpc:DescribeVpcs",
        "vpc:DescribeVSwitches"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ecs:CreateSecurityGroup",
        "ecs:AuthorizeSecurityGroup",
        "ecs:CreateNetworkInterface",
        "ecs:DescribeNetworkInterfaces",
        "ecs:AttachNetworkInterface",
        "ecs:DescribeNetworkInterfacePermissions",
        "ecs:CreateNetworkInterfacePermission"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "oss:*",
      "Resource": "*",
      "Effect": "Allow"
    }
```

```
    ]
}
```

Attach a policy to a RAM role

After you create a RAM role, you can attach a specific policy to the RAM role.

1. Log on to the RAM console.
   - Log on to the RAM console by using your Alibaba Cloud account.
   - Log on to the RAM console as a RAM user.
2. In the left-side navigation pane, choose **Permissions > Policies**.
3. On the Policies page, click **Create Policy**.
4. On the Create Policy page, configure **Name** and **Note**. In this example, the policy name is **AliyunStreamDefaultRolePolicy**.
5. In the code editor below **Policy Document**, enter the following code and click **OK**:

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "vpc:DescribeVpcs",
        "vpc:DescribeVSwitches"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ecs:CreateSecurityGroup",
        "ecs:AuthorizeSecurityGroup",
        "ecs:CreateNetworkInterface",
        "ecs:DescribeNetworkInterfaces",
        "ecs:AttachNetworkInterface",
        "ecs:DescribeNetworkInterfacePermissions",
        "ecs:CreateNetworkInterfacePermission"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

> **Note** You can delete the following permissions after you create a cluster:
> - ecs:CreateSecurityGroup
> - ecs:AuthorizeSecurityGroup

6. In the left-side navigation pane, click **Roles**. On the Roles page, find **AliyunStreamDefaultRole** in the role list and click **Add Permissions** in the **Actions** column.
7. In the Add Permissions panel, click **Custom Policy** in the **Select Policy** section and enter `AliyunOSSFullAccess in the search box below Custom Policy`.

8. Click **AliyunOSSFullAccess** in the **Authorization Policy Name** column.
9. In the **Add Permissions** panel, click **Custom Policy** in the **Select Policy** section.
0. In the search box below **Custom Policy** of the **Select Policy** section, enter `AliyunStreamDefaultRolePolicy`.
1. Click **AliyunStreamDefaultRolePolicy** in the **Authorization Policy Name** column.
2. Click **OK**.

5.Blink SQL reference

5.1. Overview

Flink SQL is a programming language that is developed by Alibaba Cloud to simplify the computing model of Realtime Compute for Apache Flink and to decrease the requirements for user skills. Flink SQL complies with standard SQL semantics.

This topic describes how to use Flink SQL in Realtime Compute for Apache Flink. The topic covers the following aspects:

- Basic concepts
- Keywords
- Data types
- DDL statements
- DML statements
- Query statements
- Data views
- Window functions
- Logical functions
- Built-in functions
- UDXs

5.2. Keywords

This topic describes the reserved keywords in Realtime Compute for Apache Flink and how to use these keywords.

Common keyword types

Common type	Keyword
Data type	• VARCHAR • INT • BIGINT • DOUBLE • DATE • BOOLEAN • TINYINT • SMALLINT • FLOAT • DECIMAL • VARBINARY
DDL	• CREATE TABLE • CREATE FUNCTION • CREATE VIEW
DML	INSERT INTO

SELECT clause	• SELECT FROM • WHERE • GROUP BY • 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 contain only letters, digits, and underscores (_).

Reserved keywords

The following combinations of strings are reserved as keywords in Realtime Compute for Apache Flink. 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,ARR
,AS,ASC,ASENSITIVE,ASSERTION,ASSIGNMENT,ASYMMETRIC,AT,ATOMIC,ATTRIBUTE,ATTRIBUTES,AUTHORI
TION,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,CE
URY,CHAIN,CHAR,CHARACTER,CHARACTERISTICTS,CHARACTERS,CHARACTER_LENGTH,CHARACTER_SET_CATAL
,CHARACTER_SET_NAME,CHARACTER_SET_SCHEMA,CHAR_LENGTH,CHECK,CLASS_ORIGIN,CLOB,CLOSE,COALES
,COBOL,COLLATE,COLLATION,COLLATION_CATALOG,COLLATION_NAME,COLLATION_SCHEMA,COLLECT,COLUMN
OLUMN_NAME,COMMAND_FUNCTION,COMMAND_FUNCTION_CODE,COMMIT,COMMITTED,CONDITION,CONDITION_NU
ER,CONNECT,CONNECTION,CONNECTION_NAME,CONSTRAINT,CONSTRAINTS,CONSTRAINT_CATALOG,CONSTRAIN
NAME,CONSTRAINT_SCHEMA,CONSTRUCTOR,CONTAINS,CONTINUE,CONVERT,CORR,CORRESPONDING,COUNT,COV
_POP,COVAR_SAMP,CREATE,CROSS,CUBE,CUME_DIST,CURRENT,CURRENT_CATALOG,CURRENT_DATE,CURRENT_
FAULT_TRANSFORM_GROUP,CURRENT_PATH,CURRENT_ROLE,CURRENT_SCHEMA,CURRENT_TIME,CURRENT_TIMES
MP,CURRENT_TRANSFORM_GROUP_FOR_TYPE,CURRENT_USER,CURSOR,CURSOR_NAME,CYCLE,
DATA,DATABASE,DATE,DATETIME_INTERVAL_CODE,DATETIME_INTERVAL_PRECISION,DAY,DEALLOCATE,DEC,
CADE,DECIMAL,DECLARE,DEFAULT,DEFAULTS,DEFERRABLE,DEFERRED,DEFINED,DEFINER,DEGREE,DELETE,D
SE_RANK,DEPTH,DEREF,DERIVED,DESC,DESCRIBE,DESCRIPTION,DESCRIPTOR,DETERMINISTIC,DIAGNOSTIC
DISALLOW,DISCONNECT,DISPATCH,DISTINCT,DOMAIN,DOUBLE,DOW,DOY,DROP,DYNAMIC,DYNAMIC_FUNCTION
YNAMIC_FUNCTION_CODE,
EACH,ELEMENT,ELSE,END,END-
EXEC,EPOCH,EQUALS,ESCAPE,EVERY,EXCEPT,EXCEPTION,EXCLUDE,EXCLUDING,EXEC,EXECUTE,EXISTS,EXP
XPLAIN,EXTEND,EXTERNAL,EXTRACT,
FALSE,FETCH,FILTER,FINAL,FIRST,FIRST_VALUE,FLOAT,FLOOR,FOLLOWING,FOR,FOREIGN,FORTRAN,FOUN
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,
STANTIABLE,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
N,LOCAL,LOCALTIME,LOCALTIMESTAMP,LOCATOR,LOWER,
M,MAP,MATCH,MATCHED,MAX,MAXVALUE,MEMBER,MERGE,MESSAGE_LENGTH,MESSAGE_OCTET_LENGTH,MESSAGE
EXT,METHOD,MICROSECOND,MILLENNIUM,MIN,MINUTE,MINVALUE,MOD,MODIFIES,MODULE,MONTH,MORE,MULT
ET,MUMPS,
NAME,NAMES,NATIONAL,NATURAL,NCHAR,NCLOB,NESTING,NEW,NEXT,NO,NONE,NORMALIZE,NORMALIZED,NOT
ULL,NULLABLE,NULLIF,NULLS,NUMBER,NUMERIC,
OBJECT,OCTETS,OCTET_LENGTH,OF,OFFSET,OLD,ON,ONLY,OPEN,OPTION,OPTIONS,OR,ORDER,ORDERING,OR
```

```
NALITY,OTHERS,OUT,OUTER,OUTPUT,OVER,OVERLAPS,OVERLAY,OVERRIDING,
PAD,PARAMETER,PARAMETER_MODE,PARAMETER_NAME,PARAMETER_ORDINAL_POSITION,PARAMETER_SPECIFIC
ATALOG,PARAMETER_SPECIFIC_NAME,PARAMETER_SPECIFIC_SCHEMA,PARTIAL,PARTITION,PASCAL,PASSTHR
GH,PATH,PERCENTILE_CONT,PERCENTILE_DISC,PERCENT_RANK,PLACING,PLAN,PLI,POSITION,POWER,PREC
ING,PRECISION,PREPARE,PRESERVE,PRIMARY,PRIOR,PRIVILEGES,PROCEDURE,PUBLIC,
QUARTER,
RANGE,RANK,READ,READS,REAL,RECURSIVE,REF,REFERENCES,REFERENCING,REGR_AVGX,REGR_AVGY,REGR_
UNT,REGR_INTERCEPT,REGR_R2,REGR_SLOPE,REGR_SXX,REGR_SXY,REGR_SYY,RELATIVE,RELEASE,REPEATA
E,RESET,RESTART,RESTRICT,RESULT,RETURN,RETURNED_CARDINALITY,RETURNED_LENGTH,RETURNED_OCTE
LENGTH,RETURNED_SQLSTATE,RETURNS,REVOKE,RIGHT,ROLE,ROLLBACK,ROLLUP,ROUTINE,ROUTINE_CATALO
ROUTINE_NAME,ROUTINE_SCHEMA,ROW,ROWS,ROW_COUNT,ROW_NUMBER,
SAVEPOINT,SCALE,SCHEMA,SCHEMA_NAME,SCOPE,SCOPE_CATALOGS,SCOPE_NAME,SCOPE_SCHEMA,SCROLL,SE
CH,SECOND,SECTION,SECURITY,SELECT,SELF,SENSITIVE,SEQUENCE,SERIALIZABLE,SERVER,SERVER_NAME
ESSION,SESSION_USER,SET,SETS,SIMILAR,SIMPLE,SIZE,SMALLINT,SOME,SOURCE,SPACE,SPECIFIC,SPEC
ICTYPE,SPECIFIC_NAME,SQL,SQLEXCEPTION,SQLSTATE,SQLWARNING,SQL_TSI_DAY,SQL_TSI_FRAC_SECOND
QL_TSI_HOUR,SQL_TSI_MICROSECOND,SQL_TSI_MINUTE,SQL_TSI_MONTH,SQL_TSI_QUARTER,SQL_TSI_SECO
,SQL_TSI_WEEK,SQL_TSI_YEAR,SQRT,START,STATE,STATEMENT,STATIC,STDDEV_POP,STDDEV_SAMP,STREA
STRUCTURE,STYLE,SUBCLASS_ORIGIN,SUBMULTISET,SUBSTITUTE,SUBSTRING,SUM,SYMMETRIC,SYSTEM,SYS
M_USER,
TABLE,TABLESAMPLE,TABLE_NAME,TEMPORARY,THEN,TIES,TIME,TIMESTAMP,TIMESTAMPADD,TIMESTAMPDIF
TIMEZONE_HOUR,TIMEZONE_MINUTE,TINYINT,TO,TOP_LEVEL_COUNT,TRAILING,TRANSACTION,TRANSACTION
ACTIVE,TRANSACTIONS_COMMITTED,TRANSACTIONS_ROLLED_BACK,TRANSFORM,TRANSFORMS,TRANSLATE,TRA
LATION,TREAT,TRIGGER,TRIGGER_CATALOG,TRIGGER_NAME,TRIGGER_SCHEMA,TRIM,TRUE,TYPE,
UESCAPE,UNBOUNDED,UNCOMMITTED,UNDER,UNION,UNIQUE,UNKNOWN,UNNAMED,UNNEST,UPDATE,UPPER,UPSE
,USAGE,USER,USER_DEFINED_TYPE_CATALOG,USER_DEFINED_TYPE_CODE,USER_DEFINED_TYPE_NAME,USER_
FINED_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
```

5.3. Basic concepts

5.3.1. Time zone

This topic describes how to configure a time zone for a Realtime Compute for Apache Flink job to adjust the output data of the DATE and TIME types.

Configure a time zone

- Configure a time zone
  You can configure a time zone for a Realtime Compute for Apache Flink job, such as `blink.job.timeZone=America/New_York`, to adjust the output data of the DATE and TIME types. The default time zone is UTC+8.
  - For more information about time zones, see Supported time zones.
  - For more information about how to configure a time zone, see Specify job parameters.
- Configure different time zones for different source or result tables
  You can configure different time zones for different source or result tables. For example, if you want to read or write data of the TIME, DATE, or TIMESTAMP type in a MySQL database that uses the `America/New_York` time zone, but the `Asia/Shanghai` time zone is used in the computation of a job, you can set a time zone for the source or result table separately

in the following way:

```
CREATE TABLE mysql_source_my_table (
 -- ...
) WITH (
timeZone='America/New_York'
 -- ...
);
```

Examples

In Realtime Compute for Apache Flink, the time zones that are used in time zone-related functions are custom time zones. The custom time zone `Asia/Shanghai` is used in the following examples.

- Functions that convert values from the STRING type to the TIMESTAMP type: TO_TIMESTAMP, TIMESTAMP, and UNIX_TIMESTAMP

```
TO_TIMESTAMP('2020-08-03 10:59:45.957')
-- The output is `2020-08-03 10:59:45.957`.

TIMESTAMP '2020-08-03 10:59:45.957'
-- The output is `2020-08-03 10:59:45.957`.

UNIX_TIMESTAMP('2020-08-03 10:59:45.957')
-- The output is `1596423585`.
```

- Functions that convert values from the TIMESTAMP type to the STRING type: DATE_FORMAT and FROM_UNIXTIME

> **Note** If the input parameter is of the TIMESTAMP type, the output data varies based on your custom time zone.

```
DATE_FORMAT(TO_TIMESTAMP(1596702949000), 'yyyy-MM-dd HH:mm:ss')
-- The output is `2020-08-06 16:35:49`.

DATE_FORMAT('2020-08-06 16:35:49', 'yyyy-MM-dd HH:mm:ss', 'yyyy/MM/dd HH:mm:ss')
-- The output is `2020/08/06 16:35:49`.

FROM_UNIXTIME(1596702949000/1000)
-- The output is `2020-08-06 16:35:49`.
```

- Time-related functions
  If the input parameters are of the TIMESTAMP type, the output data of the functions such as EXTRACT, FLOOR, CEIL, and DATE_DIFF varies based on your custom time zone.

```
-- 1521503999000 2018-03-19T23:59:59+0000, 2018-03-20T07:59:59+0800
 EXTRACT(DAY FROM TO_TIMESTAMP(1521503999000))
-- The output is `20`. This value indicates the twentieth day of the month in the UTC
+8 time zone.
```

- Functions used to calculate the current time
  Functions used to calculate the current time, including `LOCALTIMESTAMP()`, `CURRENT_TIMESTAMP()`, `NOW()`, and `UNIX_TIMESTAMP()`

```
-- The current time is 2020-08-03 10:59:45 in the Asia/Shanghai time zone.

LOCALTIMESTAMP
-- The output is `2020-08-03 10:59:45.957`.

CURRENT_TIMESTAMP
-- The output is `2020-08-03 10:59:45.957`.

NOW()
-- The output is `1596423585`.

UNIX_TIMESTAMP()
-- The output is `1596423585`.
```

- Functions used to calculate data of the DATE and TIME types

  In Flink SQL, data of the DATE and TIME types is expressed and calculated as integers. A DATE value refers to the number of days that have elapsed after 00:00:00 Thursday, 1 January 1970. A TIME value refers to the number of milliseconds that have elapsed after 00:00:00 on the current day of your time zone. If data of the DATE and TIME types is calculated in user-defined functions (UDFs), a time zone offset is added to the Java object when data of the INT type is converted to the `java.sql.Date` or `java.sql.Time` type.

Supported time zones

For more information about the time zones that are supported by Realtime Compute for Apache Flink, see Time zones.

5.3.2. Time attributes

This topic describes the following time attributes that are supported by Blink SQL: event time and processing time.

Apache Flink supports three time attributes for the processing of streaming data: processing time, event time, and ingestion time.Blink SQL supports only two of the three time attributes:

- Event time: the event time that you provide in the data store. In most cases, the event time is the original time when the data is created.
- Processing time: the local system time when the system processes an event. The unit is milliseconds.

Event Time

The event time is also known as rowtime. The event time attribute must be declared in the data definition language (DDL) statement that you execute to create a source table. You can declare a field in the source table as the rowtime field. Note that you can declare a field of only the TIMESTAMP type as the rowtime field. In the future, you can declare a field of the LONG type as the rowtime field. If the source table does not contain a TIMESTAMP column, you can use a computed column to create a TIMESTAMP column based on an existing column. For more information, see Computed column.

In some scenarios, the order in which data records are received may be different from the order in which they are processed. The possible causes include out-of-order input data and network jitters. The network jitters may be caused by network congestions and transmission latencies. Before you define a rowtime field, define a computing method for watermarks in an explicit way. For more information, see Watermark.

In the following example, data is aggregated by using event time-based window functions:

```
CREATE TABLE tt_stream (
  a VARCHAR,
  b VARCHAR,
  ts TIMESTAMP,
  WATERMARK wk1 FOR ts as withOffset (ts, 1000) --Define a computing method for waterma
rks.
) WITH (
  type = 'sls',
  topic = '<yourTopicName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>'
);
CREATE TABLE rds_output (
  id VARCHAR,
  win_start TIMESTAMP,
  win_end 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 (ts, INTERVAL '1' SECOND) AS win_start,
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,
  COUNT (a) AS cnt
FROM
  tt_stream
GROUP
  BY SESSION (ts, INTERVAL '1' SECOND),
  a
```

Processing Time

The processing time is generated by the system and is not included in the raw data. Therefore, you must explicitly define a processing time column when you declare the source table.

```
filedName as PROCTIME()
```

In the following example, processing time-based window functions are used to aggregate data:

```
CREATE TABLE mq_stream (
    a VARCHAR,
    b VARCHAR,
    c BIGINT,
    ts AS PROCTIME () --Explicitly define a processing time column when you declare the
source table.
  ) WITH (
    type = 'mq',
    topic = '<yourTopic>',
    accessId = '<yourAccessId>',
    accessKey = '<yourAccessSecret>'
  );
CREATE TABLE rds_output (
  id VARCHAR,
  win_start TIMESTAMP,
  win_end TIMESTAMP,
  cnt BIGINT
) with (
  type = 'rds',
  url = '<yourDatebaseURL>',
  tableName = '<yourDatabasTableName>',
  userName = '<yourUserName>',
  password = '<yourPassword>'
);
INSERT
  INTO rds_output
SELECT
  a AS id,
  SESSION_START (ts, INTERVAL '1' SECOND) AS win_start,
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,
  COUNT (a) AS cnt
FROM
  mq_stream
GROUP
  BY SESSION (ts, INTERVAL '1' SECOND),
  a
```

Expiration of time attributes

The time attribute of fields no longer takes effect after one of the following operations is completed:

- GROUP BY operations on the fields that are not defined as time attribute fields, except the GROUP BY operations in tumbling, sliding, and session windows. For more information, see TUMBLE, HOP, and SESSION.
- JOIN operations on two data streams.

  > **Note** For more information, see JOIN statements.

- MATCH_RECOGNIZE operations in complex event processing (CEP) statements. For more information, see CEP statements.
- PARTITION BY operations in OVER windows. For more information, see OVER windows.
- UNION operations. **UNION** is equivalent to the combination of RETRACT and UNION ALL.

If you use time-based window functions for computing after the preceding operations, errors

are returned, such as `org.apache.flink.table.api.ValidationException: Window can only be defined over a time attribute column.`

5.3.3. Watermark

Realtime Compute for Apache Flink aggregates data by using window functions based on the time attribute. To use window functions based on the event time of a job, you must define a watermark when you declare a source table.

A watermark is used to measure the progress of Event Time. It is a hidden data attribute. A watermark is defined in the DDL statement of the source table. Flink provides the following statement to define a watermark:

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

> **Note** For more information about the time attributes of Realtime Compute for Apache Flink, see Time attributes.

Parameter	Required	Description
watermarkName	No	The name of the watermark.
<rowtime_field>	Yes	The column used to generate the watermark. <rowtime_field> is identified as the event time column and must be a column of the TIMESTAMP type defined in the table. You can use <rowtime_field> to define a window in the job code.
withOffset	Yes	The policy to generate a watermark. In this example, the watermark value is generated by using the following formula: `<rowtime_field> - offset`. The first parameter in `withOffset` must be <rowtime_field>.
offset	Yes	The offset between the watermark value and event time, in milliseconds.

A specific field in a data record indicates the time when the record was generated. For example, a table contains a `rowtime` field whose data type is TIMESTAMP, and one of its values is `1501750584000 (2017-08-03 08:56:24.000)`. If you want to define a watermark based on the `rowtime` field and configure a 4-second offset, add the following definition:

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

In this example, the watermark time of the data record is `1501750584000 - 4000 = 1501750580000 (2017-08-03 08:56:20.000)`. This means that all data whose timestamp is earlier than `1501750580000 (2017-08-03 08:56:20.000)` has arrived.

> **Note**
> - If you use an event time-based watermark, the `rowtime` field must be of the TIMESTAMP type. Realtime Compute for Apache Flink supports 13-digit UNIX timestamps in milliseconds. If the rowtime field is of another type or the UNIX timestamp is not 13 digits in length, we recommend that you use a computed column to convert the time. For more information, see Computed column.
> - The event time and processing time can only be declared in the source table. For more information, see Event Time and Processing Time.

Summary

- A watermark indicates that all the events whose timestamp t' is earlier than the watermark time t (`t'< t`) have occurred. After the watermark time `t` takes effect, all subsequently received data records whose event time is earlier than `t` are discarded. Realtime Compute for Apache Flink will allow you to change the configuration and update the subsequent data.
- Watermarks are important for data streams that arrive out of order because the watermarks help ensure that the computing in a window is correct even if some events arrive late.
- If an operator has multiple input data streams for parallel processing, the event time of the data stream with the shortest time is used as the event time of the operator.

5.3.4. Computed column

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

Concept

A computed column is a virtual column that is not stored in a physical table. You can create computed columns by using expressions, built-in functions, or user-defined extensions (UDXs). In Flink SQL, a computed column can be used the same as columns that are stored in a physical table.

Usage

Currently, the event time (also known as rowtime) column in a Watermark must be of the TIMESTAMP type. The LONG data type will be supported in the future. 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 type, you can use a computed column to convert a field of another type to the TIMESTAMP type.

Syntax

```
column_name AS computed_column_expression
```

Example

The rowtime column in a watermark must be of the TIMESTAMP type. Currently, Realtime Compute only supports 13-bit UNIX timestamps measured in milliseconds. If the TIME column in a DataHub source table is defined as a 16-bit UNIX timestamp measured in microseconds, you can use a computed column to convert the 16-bit UNIX timestamp to a 13-bit UNIX timestamp. The sample code is as follows:

```
CREATE TABLE test_stream(
  a INT,
  b BIGINT,
  `TIME` BIGINT,
  ts AS TO_TIMESTAMP(`TIME`/1000), -- Use a computed column to convert 16-bit timestamps to 13-bit timestamps.
  WATERMARK FOR ts AS WITHOFFSET(ts, 1000)
) WITH (
  type = 'datahub',
  ...
);
```

The `` `TIME` `` field in the source table contains the date and time information. The value of this field is of the BIGINT type. A computed column is created to convert the `TIME` column of the BIGINT type to the `ts` column of the TIMESTAMP type. The `ts` column is used as the rowtime of a watermark.

5.4. Data types

5.4.1. Data type conversion

This topic describes the data types that are supported by Realtime Compute for Apache Flink and how to convert between different data types.

Data types supported by Realtime Compute for Apache Flink

Data type	Description	Value range
VARCHAR	Stores strings of varying length.	A VARCHAR string can store a maximum of 4 MB of data.
BOOLEAN	Stores logical values.	Valid values: TRUE, FALSE, and UNKNOWN.
TINYINT	Stores tiny integers. Each tiny integer occupies 1 byte.	`-128 to 127`.
SMALLINT	Stores small integers. Each small integer occupies 2 bytes.	`-32768 to 32767`.
INT	Stores integers. Each integer occupies 4 bytes.	`-2147483648 to 2147483647`.
BIGINT	Stores big integers. Each big integer occupies 8 bytes.	`-9223372036854775808 to 9223372036854775807`.
FLOAT	Stores single-precision floating-point numbers. Each single-precision floating-point number occupies 4 bytes.	Each single-precision floating-point number is accurate to six decimal places.

DECIMAL	Stores the numbers that have the fixed precision and scale. The precision specifies the total number of digits both to the left and to the right of the decimal point. The scale specifies the number of digits to the right of the decimal point.	For example, the value of `DECIMAL(5,2)` is `123.45`.
DOUBLE	Stores double-precision floating-point numbers. Each double-precision floating-point number occupies 8 bytes.	Each double-precision floating-point number is accurate to 15 decimal places.
DATE	Stores dates.	Example value: `DATE'1969-07-20'`.
TIME	Stores time.	Example value: `TIME '20:17:40'`.
TIMESTAMP	Stores timestamps. Each timestamp contains both date and time.	Example value: `TIMESTAMP '1969-07-20 20:17:40'`.
VARBINARY	Stores binary data.	This type corresponds to the `byte[]` array. **Note** The storage size for the VARBINARY data type is not limited.

Data type conversion

Convertible	Data Types										
	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

Example

- Test data

var1 (VARCHAR)	big1 (BIGINT)
1000	323

- Test statements

```
cast (var1 as bigint) as AA;
cast (big1 as varchar) as BB;
```

- Test results

AA (BIGINT)	BB (VARCHAR)
1000	323

5.4.2. Mathematical and logical operations between data types

This topic describes mathematical and logical operations between different data types in Realtime Compute for Apache Flink.

> **Note** The data types of numeric1 and numeric2 in a mathematical operation must be the same.

Mathematical operation	Description	Data types supported by numeric1 and numeric2	Example
numeric1 + numeric2	Returns the sum of two numbers in a mathematical operation.	• INT • DOUBLE • DECIMAL • BIGINT	`2+4.2`
numeric1 - numeric2	Returns the difference between two numbers in a mathematical operation.		`3-5.3`
numeric1 * numeric2	Returns the product of two numbers in a mathematical operation.		`2*4`
numeric1 / numeric2	Returns the quotient of two numbers in a mathematical operation.		`2.4/5`
numeric1 > numeric2	Checks whether the first number is greater than the second number in a mathematical operation.		`2.4>5`
numeric1 < numeric2	Checks whether the first number is less than the second number in a mathematical operation.		`2.4<5`
numeric1 >= numeric2	Checks whether the first number is greater than or equal to the second number in a mathematical operation.		`2.4>=5`
numeric1 <= numeric2	Checks whether the first number is less than or equal to the second number in a mathematical operation.		`2.4<=5`

numeric1 = numeric2	Checks whether the two numbers in a mathematical operation are equal to each other.	• INT • DOUBLE • DECIMAL • BIGINT • VARCHAR	`'iphone' = 5`
numeric1 <> numeric2	Checks whether the two numbers in a mathematical operation are not equal to each other.		`'iphone' <> 5`

5.5. Create a data view

This topic describes how to create a data view in Realtime Compute for Apache Flink to simplify the development process.

Background information

If your business logic is complex, you must write nested statements in a DML statement, which makes it difficult to locate a problem. To simplify the development process, you can define a data view and write nested statements to the data view.

> **Note** A data view displays a logical table that describes the computing logic. It does not physically store data.

Syntax

```
CREATE VIEW viewName[ (columnName[ , columnName]* ) ] AS queryStatement;
```

- viewName: the name of the view.
- columnName: the name of the field.
- queryStatement: the alias of the nested statement.

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

- Test data

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

- Test statements

```
CREATE TABLE datahub_stream (
   a VARCHAR,
   b BIGINT,
   c TIMESTAMP,
   d AS PROCTIME()
) WITH (
  TYPE='datahub',
  ...
);

CREATE TABLE rds_output (
   a VARCHAR,
   b TIMESTAMP,
   cnt BIGINT,
   PRIMARY KEY(a)
)WITH(
  TYPE = 'rds',
  ...
);

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;
```

- Test results

a(VARCHAR)	b (TIMESTAMP)	cnt (BIGINT)
test2	2017-11-06 16:54:10	4

5.6. DDL statements

5.6.1. Overview

This topic describes the data definition language (DDL) syntax that is supported by Realtime Compute for Apache Flink, and the field mapping and case sensitivity issues that require your attention when you use DDL.

Syntax

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

Description

Realtime Compute for Apache Flink does not store data. All the DDL statements that are executed to create tables declare references to external tables and data stores.

```
CREATE TABLE mq_stream(
 a VARCHAR,
 b VARCHAR,
 c VARCHAR
) WITH (
 type='mq',
 topic='blink_mq_test',
 accessID='<yourAccessID>',
 accessKey='<yourAccessSecret>'
);
```

The preceding code does not create a topic of the Message Queue source table in Flink SQL. Instead, the code is used to declare a reference to the `mq_stream` table. For all the downstream data manipulation language (DML) operations on the Message Queue topic `blink_mq_test`, you can replace the topic name with the alias `mq_stream`. When you declare references to external tables, pay attention to the following points:

- In Realtime Compute for Apache Flink, a declaration of a table is valid only for the current job. A Realtime Compute for Apache Flink job is generated after you submit an SQL file. Therefore, the preceding declaration related to the `mq_stream` table is valid only for the current SQL file. Different SQL files in the same Realtime Compute for Apache Flink project can declare reference of the `mq_stream` table.
- Based on the standard SQL definitions, keywords, table names, and column names in DDL statements are not case-sensitive.
- The names of tables and columns must start with a letter, and can contain only letters, digits, and underscores (_).
- DDL declarations may establish the field mappings between the declaration table and the external table based on field names or other factors. This depends on the nature of the upstream plug-in that is used. To prevent data errors caused by inaccurate definitions, we recommend that you use the same field names and field quantity in your declaration as those in the referenced external tables.

  > **Note**
  > - If upstream and downstream plug-ins support retrieving values based on keys, the declared table and its referenced external table can have different field quantity. However, the field names must be the same.
  > - If the upstream and downstream plug-ins do not support retrieving values based on keys, the declaration table and its referenced external table must have the same number and sequence of fields.

Field mapping

A declared table supports the following two field mapping methods based on whether its external data source has a schema:

- Sequence mapping

  This method applies to data sources that have no schema, such as Message Queue. These data sources are usually unstructured storage systems that do not support retrieving values based on keys. We recommend that you customize field names in DDL SQL statements and use the same field types and field quantity in the declared table as those in the external table.

  The following record in Message Queue is used as an example:

  ```
  asavfa,sddd32,sdfds
  ```

  Specify Message Queue field names based on the naming conventions.

  ```
  CREATE TABLE mq_stream(
   a VARCHAR,
   b VARCHAR,
   c VARCHAR
  ) WITH (
   type='mq',
   topic='blink_mq_test',
   accessID='<yourAccessID>',
   accessKey='<yourAccessSecret>'
  );
  ```

- Name mapping
  This method applies to data sources that have a schema. These data sources define field names and field types at the table storage level and support retrieving values based on keys. We recommend that you use the same schema definitions in Flink SQL declarations as those of the external storage system. Specifically, the names, number, and sequence of fields in the declaration table must be the same as those in the external table.

  > **Note** If field names in an external storage system such as Tablestore are case-sensitive, you must enclose the field names in grave accents (`). In the DDL syntax, field names in the declaration table must be the same as those in the external table.

Case sensitivity

Realtime Compute for Apache Flink adopts standard SQL statements. Therefore, fields are not case-sensitive. For example, the following two statements have the same meaning:

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

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

However, fields in some external data sources that are referenced by Realtime Compute for Apache Flink are case-sensitive, such as Tablestore. The following statement defines the

uppercase `NAME` field for Tablestore.

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

In all subsequent DML statements, enclose the field in grave accents (`) if the field is referenced. In the following example, the sample code is provided.

```
INSERT INTO tableA
SELECT
  `NAME`,
  `VALUE`
FROM
  tableB;
```

References

For more information about how to create source tables, dimension tables, and result tables in Realtime Compute for Apache Flink, see the following topics:

- Overview of source tables
- Overview of result tables
- Overview

5.6.2. Create a source table

5.6.2.1. Overview of source tables

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

Syntax

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

Example

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

Obtain attribute fields of a source table

- **Syntax**

  Realtime Compute provides the `HEADER` keyword in the DDL statement of a source table for you to obtain the attribute fields from the source table.

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

  In this example, the `'timestamp'` field is defined as `HEADER`. Realtime Compute reads the values of attribute fields from the source table. Then, the 'timestamp' field is used as a common field.

> **Note** The default attribute fields vary depending on the source table type, such as DataHub, Log Service, and Message Queue (MQ). You can customize attribute fields for certain types of source tables. For more information, see the topics about source tables of the related type.

- **Example**

  The following table uses a source table of Log Service to describe how to obtain attribute fields of the source table. Currently, a source table of Log Service has three attribute fields listed in the following table.

Field name	Description
`__source__`	The message source.
`__topic__`	The message topic.
`__timestamp__`	The time when a log was generated.

> **Note** To obtain attribute fields of a source table, you must add the `HEADER` keyword to the end of a field declaration.

The example is as follows:

- Test data

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

- Test statements

```
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
```

- Test results

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

Source tables with window functions

Realtime Compute aggregates data in windows based on two time attributes: event time and processing time. For more information, see Event Time and Processing Time. If window functions are used in a Realtime Compute job, you must define a watermark and computed column in the DDL statement of a source table. For more information, see Watermark and Computed column. For more information about data aggregation based on time attributes in Realtime Compute, see Time attributes.

Supported source table types

Realtime Compute allows you to create multiple types of source tables. For more information, see the following topics:

- Create a Log Service source table
- Create a ApsaraMQ for RocketMQ source table
- Create a Message Queue for Apache Kafka source table
- Create a Tablestore source table

- Create a full MaxCompute source table

5.6.2.2. Create an Oracle database source table

This topic describes how to create an Oracle database source table. It also describes the parameters in the WITH clause, data type mapping, sample code, and FAQ involved when you create an Oracle database source table.

> **Important**
> - This topic applies only to Blink 3.4.X and later.
> - You can create Oracle database source tables only when you use Oracle 11g.
> - You cannot change the number of concurrent jobs for an Oracle database source table. By default, only one job is allowed for a source table.

Syntax

In Realtime Compute for Apache Flink, you can use an Oracle database to store input data. The following code shows an example:

```
create table oracle_source (
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type = 'oracle',
    url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',
    userName = 'userName',
    password = 'password',
    dbName = 'hr',
    tableName = 'job_history',
    timeField = 'START_DATE',
    startTime = '2007-1-1 00:00:00'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to oracle.
url	The connection string of the database.	Yes	The value of this parameter is in the `jdbc:oracle:thin:@//Database IP address:Port number/Database name` format. Example: jdbc:oracle:thin:@//127.0.0.1:1521/XE.
userName	The username that is used to log on to the database.	Yes	None.
password	The password that is used to log on to the database.	Yes	None.

tableName	The name of the table in the database. Database table names can be in one of the following formats: • Table name 1,Table name 2 • Database name.Table name 1,Table name 2 **Note** Multiple table names are separated by commas (,).	Yes	• table1,table2 • db1.table1,table2
timeField	The time when the database was updated.	Yes	None.
dbName	The database name.	No	If you have specified the **tableName** parameter, you do not need to specify the **dbName** parameter.
startTime	The start time of reading data from the source table.	No	2019-5-15 00:00:00
timeZone	The time zone of the database.	No	Asia/Shanghai", "UTC
queryTimeRangeMs	The time that is taken for data retrieval. Unit: milliseconds. **Note** The value of the **queryTimeRangeMs** parameter must be greater than the value of the **queryIntervalMs** parameter.	No	Default value: 5000.
queryIntervalMs	The interval at which data is queried from the database. Unit: milliseconds.	No	Default value: 100.
connectionMaxActive	The maximum number of active connections.	No	Default value: 10.
maxRetry	The maximum number of retries upon a connection failure.	No	Default value: 3.
escapeFields	Specifies whether to escape field names in the database.	No	Valid values: • false: not case-sensitive. This is the default value. • true: case-sensitive.

lengthCheck	The policy for checking the number of fields that are parsed from a row of data.	No	Valid values: • NONE: This is the default value. ◦ If the number of fields parsed from a row of data is greater than the defined number of fields, data is extracted from left to right based on the order of defined fields. ◦ If the number of fields parsed from a row of data is less than the defined number of fields, the current row is skipped. • SKIP: If the number of fields parsed from a row of data is different from the defined number of fields, the current row is skipped. • EXCEPTION: If the number of fields parsed from a row of data is different from the defined number of fields, an error is returned. • PAD: ◦ If the number of fields parsed from a row of data is greater than the defined number of fields, data is padded from left to right based on the order of defined fields. ◦ If the number of fields parsed from a row of data is less than the defined number of fields, the values of the missing fields are padded with null from left to right.
columnErrorDebug	Specifies whether the debugging feature is enabled. **Note** If the debugging feature is enabled, the system displays the logs that record the parse failures.	No	Default value: false.

Mapping between field data types

Data type of the Oracle database	Data type of Realtime Compute for Apache Flink
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

Sample code

The following example shows how to create an Oracle database source table in a Realtime Compute for Apache Flink job.

```
create table oracle_source (
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type = 'oracle',
    url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',
    userName = 'userName',
    password = 'password',
    dbName = 'hr',
    tableName = 'job_history',
    timeField = 'START_DATE',
    startTime = '2007-1-1 00:00:00'
);

create table test_out(
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type='print'
);

INSERT INTO test_out
SELECT
    EMPLOYEE_ID,
    START_DATE,
    END_DATE,
    JOB_ID,
    DEPARTMENT_ID
from oracle_source;
```

FAQ

Q: What do I do if no data is found?

A: The data cannot be found because Blink is faulty. To check whether Blink is faulty, view the `Round start:[{}], end:[{}]` and `Round read records` logs on the TaskManager tab. If the logs do not contain data, Blink is faulty.

> **Note**
>
> - `Round start:[{}], end:[{}]` : displays the start time of the queried data.
> - `Round read records` : displays the queried data records.

5.6.2.3. Create a Hologres source table

This topic describes how to create a Hologres source table. It also describes data definition language (DDL) syntax, parameters in the WITH clause, data type mapping, and sample code used when you create a Hologres source table.

Limits

This topic applies only to Blink 3.6.0 and later. If you use Blink 3.6.0 or its earlier version, we recommend that you update your Blink version to 3.7.0 or later.

Usage notes

- You can use Hologres source tables to process streaming data and batch data.
- Hologres source tables support projection pushdown. This allows you to read data only from the required columns in the Hologres source tables.
- Blink jobs execute snapshot statements to read existing data from Hologres source tables at a high rate. After the read operation is complete, the jobs end. If the jobs fail to read data, they try to read the data again.
- Parallel Blink jobs can read data from one or more Hologres shards. We recommend that the number of parallel Blink jobs be no more than the number of Hologres shards.
- If you want to use Realtime Compute for Apache Flink to consume data of Hologres source tables in real time, you must enable the binary logging feature. For more information about how to enable the binary logging feature, see Subscribe to Hologres binary logs.

Introduction to Hologres

Hologres is compatible with the PostgreSQL protocol and closely connected to the big data ecosystem. Hologres allows you to analyze and process petabytes of data in high parallelism and low latency scenarios. Hologres provides an easy method for you to use the existing business intelligence (BI) tools to perform multidimensional analysis and explore your business.

DDL syntax

```
create table mysource(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='...',
  tablename='...',
  username='...',
  password='...',
  endpoint='...',
  field_delimiter='...' -- This parameter is optional.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks

type	The type of the source table.	Yes	Set the value to hologres.
dbname	The name of the database.	Yes	N/A.
tablename	The name of the table. **Note** If the public schema is not used, you must set the tableName parameter to **schema.tableName**.	Yes	N/A.
username	The username that is used to log on to the database. You must enter the AccessKey ID of your Alibaba Cloud account.	Yes	N/A.
password	The password that is used to log on to the database. You must enter the AccessKey secret of your Alibaba Cloud account.	Yes	N/A.
endpoint	The endpoint of Hologres.	Yes	For more information, see Endpoints for connecting to Hologres.
field_delimiter	The delimiter used between rows when data is being exported. **Important** Delimiters cannot be inserted into the data.	Yes	Default value: "\u0002".
bulkread	Specifies whether to read full data from a column-oriented table.	No	Valid values: • true: Realtime Compute for Apache Flink reads full data from a column-oriented table. • false: Realtime Compute for Apache Flink does not read full data from a column-oriented table.

Data data mapping

Hologres	BLINK
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

Sample code

The following sample code shows how to create a Hologres source table in a Realtime Compute for Apache Flink job.

```
create table mysource(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='...',
  tablename='...',
  username='...',
  password='...',
  endpoint='...',
  field_delimiter='...' -- This parameter is optional.
);


create table print_output(
 a varchar,
 b BIGINT,
 c BIGINT
) with (
  type='print'
);


INSERT INTO print_output
SELECT
  a, b, c
from mysource;
```

5.6.2.4. Create a Log Service source table

This topic describes how to create a Log Service source table in Realtime Compute for Apache Flink. This topic also describes the attribute fields, parameters in the WITH clause, and data type mappings used when you create a Log Service source table.

> **Important**
>
> This topic applies only to Blink 1.4.5 and later.

What is Log Service?

Log Service is an end-to-end data logging service that is developed by Alibaba Cloud. The data format of Log Service is similar to JSON. The following code shows an example:

```
{
    "a": 1000,
    "b": 1234,
    "c": "li"
}
```

Log Service stores streaming data. Therefore, Realtime Compute for Apache Flink can use Log Service tables as result tables for the processing of streaming data.

DDL syntax

The following sample code describes how to create a Log Service source table in a data definition language (DDL) statement. In the code, `sls` indicates Log Service.

```
create table sls_stream(
  a INT,
  b INT,
  c VARCHAR
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId ='<yourAccessId>',
  accessKey ='<yourAccessKey>',
  startTime = '2017-07-05 00:00:00',
  project ='<yourProjectName>',
  logStore ='<yourLogStoreName>',
  consumerGroup ='<yourConsumerGroupName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to sls.
endPoint	The endpoint of Log Service.	Yes	Endpoints.
accessId	The AccessKey ID that is used to access Log Service.	Yes	N/A.
accessKey	The AccessKey secret that is used to access Log Service.	Yes	N/A.
project	The name of the Log Service project from which data is read.	Yes	N/A.
logStore	The name of the Logstore in the Log Service project.	Yes	N/A.
startTime	The time at which logs start to be consumed.	No	N/A.

consumerGroup	The name of a consumer group.	No	You can specify this parameter based on your business requirements. The format of the name is not fixed.
heartBeatIntervalMills	The heartbeat interval of the consumer client.	No	Default value: 10000. Unit: milliseconds.
maxRetryTimes	The maximum number of retries for reading data.	No	Default value: 5.
batchGetSize	The number of log items that are read from a log group at a time.	No	Default value: 100.
columnErrorDebug	Specifies whether to enable debugging.	No	Default value: false. This indicates that debugging is disabled. If you enable debugging, logs that contain parsing exceptions are printed.

startupMode	Specifies whether to enable the consumption mode.	No	Valid values: • TIMESTAMP: Realtime Compute for Apache Flink starts to consume data from a specified time point in each shard. This is the default value. • Earliest: Realtime Compute for Apache Flink starts to consume data from the earliest offset in each shard. • Latest: Realtime Compute for Apache Flink starts to consume data from the latest offset in each shard. • Group_Offsets: Realtime Compute for Apache Flink preferentially consumes data from the checkpoint that is stored on the server in each shard. You must specify the consumerGroup parameter. The consumption mode varies based on the following scenarios: ◦ If recovery from the Flink state is successful, Realtime Compute for Apache Flink starts to consume data from the checkpoint of the Flink state. ◦ If recovery from the Flink state fails, the consumption mode varies based on the following scenarios: ▪ If a checkpoint is available for the specified consumer group, Realtime Compute for Apache Flink attempts to consume data from this checkpoint. ▪ If no checkpoint is available for the specified consumer group, the consumption mode varies based on the following scenarios: ▪ If you specify startTime, Realtime Compute for Apache Flink starts to consume data at the specified start time. ▪ If you do not specify startTime, Realtime Compute for Apache Flink starts to consume data from the earliest offset in each shard. **Important** • Only Blink 3.6.5 and later support this parameter. • The preceding configurations take effect only when no checkpoint exists in the state data.

Note

- In Realtime Compute for Apache Flink V1.6.0 and earlier, the read performance may be affected if the number of shards in a consumer group is specified. This issue is being rectified.
- Log Service does not support the MAP data type.
- Log Service sets the fields that do not exist to null.
- We recommend that you define the fields in the same order as the fields in the source table. Unordered fields are also supported.
- If input data is in the JSON format, define a separator and use the built-in function JSON_VALUE to analyze the data. Otherwise, the following parsing error is returned:

```
2017-12-25 15:24:43,467 WARN [Topology-0 (1/1)]
com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollect
- Field missing error, table column number: 3, data column number: 3, data fi
led number: 1, data:
[{"lg_order_code":"LP00000005","activity_code":"TEST_CODE1","occur_time":"2017-
12-10 00:00:01"}]
```

- The value of the **batchGetSize** parameter cannot exceed 1000. Otherwise, an error is returned.
- The **batchGetSize** parameter specifies the number of log items that are read at a time in a log group. If the value of the **batchGetSize** parameter and the size of a single log item that is specified in LogItem are large, frequent garbage collections (GCs) may occur. In this case, you must reduce the value of the batchGetSize parameter.

Data type mappings

The following table describes the mapping between the data types of Log Service and Realtime Compute for Apache Flink. We recommend that you declare the mappings in a DDL statement.

Data type of Log Service	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

Attribute fields

By default, Flink SQL supports retrieving three types of Log Service attribute fields. Custom fields are also supported as the input. For more information about how to use the attribute fields, see Obtain attribute fields of a source table.

Field	Description
`__source__`	The message source.
`__topic__`	The message topic.

`__timestamp__`	The time when a log was generated.

Sample code

```
create table sls_input(
  a int,
  b int,
  c varchar
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId ='<yourAccessI>',
  accessKey ='<yourAccessKey>',
  startTime = '2017-07-05 00:00:00',
  project ='ali-cloud-streamtest',
  logStore ='stream-test',
  consumerGroup ='consumerGroupTest1'
);

create table print_output(
 a int,
 b int,
 c varchar
) with (
  type='print'
);

INSERT INTO print_output
SELECT
  a, b, c
from sls_input;
```

FAQ

- Q: Why does the overall latency of a job increase, or why is no output generated for the job that has window aggregation?

  A: This issue occurs if no new data is written to a partition. To solve this issue, change the parallelism to be the same as the number of read and write partitions.

- Q: How do I set the parallelism?

  A: We recommend that you set the parallelism to the number of partitions. Otherwise, if the speeds at which data is read from two partitions vary significantly, data may be filtered out or data latency may occur when you set the start offset for a job to a time prior to the present time.

- Q: How do I troubleshoot the issue that the latency of a Flink job increases?

  A: The Log Service source table may be sharded. Shard indexes may not be continuous after sharding, which increases the latency of a Flink job. If you find that the latency of a Flink job increases, check whether the Log Service source table is sharded.

- Q: How do I obtain attribute fields?

  A: For more information about how to obtain attribute fields, see Obtain attribute fields of a source table.

> **Note**
>
> Attribute fields cannot be extracted during local debugging. We recommend that you perform online debugging and view the attribute fields in logs. For more information, see Online debugging.

References

- For more information about Log Service, see What is Simple Log Service?.
- For more information about how to consume Log Service data in Realtime Compute for Apache Flink, see Use Realtime Compute to consume log data.

5.6.2.5. Create a source table

This topic describes how to create a ApsaraMQ for RocketMQ source table in Realtime Compute for Apache Flink. This topic also describes the comma-separated values (CSV) file format, parameters in the WITH clause, and data type mappings used when you create a ApsaraMQ for RocketMQ source table.

> **Note**
>
> If you need to use ApsaraMQ for RocketMQ that has separate namespaces, use Blink 3.X.

Introduction to ApsaraMQ for RocketMQ

ApsaraMQ for RocketMQ is a professional messaging middleware that is developed by Alibaba Cloud. It is a core service of the enterprise-level Internet architecture. You can specify ApsaraMQ for RocketMQ tables as source tables for Realtime Compute for Apache Flink to process streaming data.

Example

```
create table mq_stream(
  x varchar,
  y varchar,
  z varchar
) with (
  type='mq',
  topic='<yourTopicName>',
  endpoint='<yourEndpoint>',
  pullIntervalMs='1000',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  startMessageOffset='1000',
  consumerGroup='<yourConsumerGroup>',
  fieldDelimiter='|'
);
```

> **Note**
>
> ApsaraMQ for RocketMQ stores unstructured data. You do not need to define schemas for ApsaraMQ for RocketMQ source tables. Instead, schemas are specified at the business layer. Realtime Compute for Apache Flink supports messages in the CSV and binary formats.

CSV format

The following example shows a ApsaraMQ for RocketMQ message in the CSV format.

```
1,name,male
2,name,female
```

> **Note**
>
> The number of data records that can be contained in a ApsaraMQ for RocketMQ message is not limited. Multiple data records are separated by `\n`.

To declare a ApsaraMQ for RocketMQ source table in a Realtime Compute for Apache Flink job, you can use the following DDL statement:

```
create table mq_stream(
  id varchar,
  name varchar,
  gender varchar
) with (
  type='mq',
  topic='<yourTopicName>',
  endpoint='<ourEndpoint>',
  pullIntervalMs='1000',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  startMessageOffset='1000',
  consumerGroup='<yourConsumerGroup>',
  fieldDelimiter='|'
);
```

Binary format

For messages in the binary format, you can use the following sample code to create a ApsaraMQ for RocketMQ source table:

```
create table source_table (
  mess varbinary
) with (
  type = 'mq',
  endpoint = '<yourEndpoint>',
  pullIntervalMs='500',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic = '<yourTopicName>',
  consumerGroup='<yourConsumerGroup>'
);

create table out_table (
  commodity varchar
) with (
  type='print'
);

INSERT INTO out_table
SELECT
  cast(mess as varchar)
FROM source_table;
```

Note

- The `cast (mess as varbinary)` method is supported only in Realtime Compute for Apache Flink that uses Blink 2.0 or later. If the Blink version is earlier than 2.0, upgrade the Blink version first.
- Data of the VARBINARY type can be passed in only once.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to mq.
topic	The name of a topic.	Yes	N/A.
			Two types of ApsaraMQ for RocketMQ services are provided: internal ApsaraMQ for RocketMQ and public ApsaraMQ for RocketMQ. Select an endpoint based on the type of ApsaraMQ for RocketMQ that you purchase.

endPoint	The endpoint of ApsaraMQ for RocketMQ.	Yes	• For jobs that run on Blink 3.7.10 or later, use Transmission Control Protocol (TCP) endpoints. For more information, see Announcement on the settings of internal TCP endpoints. You can use one of the following methods to obtain the endpoints: ◦ Internal endpoints of ApsaraMQ for RocketMQ that resides in the Alibaba Cloud classic network or a virtual private cloud (VPC): Log on to the ApsaraMQ for RocketMQ console. In the left-side navigation pane, click Instances. On the page that appears, find the instance whose endpoint you want to obtain, and click Details in the Actions column. On the Instance Details page, click the **Endpoints** tab. In the **TCP Endpoint** section, you can view the endpoint that corresponds to **Internal Access**. ◦ Public endpoint of ApsaraMQ for RocketMQ: Log on to the ApsaraMQ for RocketMQ console. In the left-side navigation pane, click Instances. On the page that appears, find the instance whose endpoint you want to obtain, and click Details in the Actions column. On the Instance Details page, click the **Endpoints** tab. In the **TCP Endpoint** section, you can view the endpoint that corresponds to **Internet Access**. • For jobs that run on Blink of a version earlier than 3.7.10, use the following endpoints: ◦ Internal endpoints of ApsaraMQ for RocketMQthat resides in the Alibaba Cloud classic network or a VPC: ▪ China (Hangzhou), China (Shanghai), China (Qingdao), China (Beijing), China (Shenzhen), and China (Hong Kong): `onsaddr-internal.aliyun.com:8080`. ▪ Singapore (Singapore): `ap-southeastaddr-internal.aliyun.com:8080`. ▪ UAE (Dubai): `ons-me-east-1-internal.aliyuncs.com:8080`. ▪ India (Mumbai): `ons-ap-south-1-internal.aliyuncs.com:8080`. ▪ Malaysia (Kuala Lumpur): `ons-ap-southeast-3-internal.aliyun.com:8080`. ◦ Public endpoint of ApsaraMQ for RocketMQ: `onsaddr-internet.aliyun.com:80`.

			Note • If you use a ApsaraMQ for RocketMQ connector in Blink of a version earlier than 3.7.10, you must update the Blink version of your Realtime Compute for Apache Flink job to Blink 3.7.10 or later and change the value of the endpoint parameter to the new endpoint of ApsaraMQ for RocketMQ. This reduces the risks of instability or unavailability that are caused by the old endpoint of ApsaraMQ for RocketMQ. For more information, see Service notices of Realtime Compute for Apache Flink. • **Internal ApsaraMQ for RocketMQ does not support cross-region access**. For example, if your Realtime Compute for Apache Flink service is located in the China (Hangzhou) region but your ApsaraMQ for RocketMQ service is located in the China (Shanghai) region, Realtime Compute for Apache Flink cannot access ApsaraMQ for RocketMQ. • The network security policies of Alibaba Cloud dynamically change. As a result, connection issues may occur when Realtime Compute for Apache Flink connects to the public ApsaraMQ for RocketMQ service. We recommend that you use the internal ApsaraMQ for RocketMQ service.
accessId	AccessKey ID	Yes	N/A.

accessKey	AccessKey Secret	Yes	N/A.
consumerGroup	The name of a consumer group.	Yes	N/A.
pullIntervalMs	The interval at which messages are pulled.	No	Unit: milliseconds.
startTime	The time when Realtime Compute for Apache Flink starts to consume messages.	No	N/A.
startMessageOffset	The start offset to consume messages.	No	This parameter is optional. If you configure this parameter, messages are consumed from the point of time that is specified by this parameter.
tag	The subscription tag.	No	N/A.
lineDelimiter	A row delimiter that is used to parse message blocks.	No	Default value: `\n`.
fieldDelimiter	A field delimiter.	No	Default value: `\u0001`. `\u0001` is the field delimiter in read-only mode, and `^A` is the field delimiter in edit mode. `\u0001` is invisible in read-only mode.
encoding	The encoding format.	No	Default value: `utf-8`.

lengthCheck	The policy that is used to check the number of fields parsed from a row of data.	No	Default value: NONE. • If the number of fields that are parsed from a row of data is greater than the specified number of fields, data is extracted from left to right based on the order of specified fields. • If the number of fields that are parsed from a row of data is less than the specified number of fields, this row of data is skipped. Other valid values are SKIP, EXCEPTION, and PAD. • SKIP: If the number of fields that are parsed from a row of data is different from the specified number of fields, this row of data is skipped. • EXCEPTION: If the number of fields that are parsed from a row of data is different from the specified number of fields, an exception is reported. • PAD: Data is padded from left to right based on the order of specified fields. ◦ If the number of fields that are parsed from a row of data is greater than the specified number of fields, data is extracted from left to right based on the order of specified fields. ◦ If the number of fields that are parsed from a row of data is less than the specified number of fields, the values of the missing fields are padded with null.
columnErrorDebug	Specifies whether to enable debugging.	No	Default value: FALSE. If you configure this parameter to TRUE, a log entry is displayed when a parsing exception occurs.
instanceID	The ID of a ApsaraMQ for RocketMQ instance.	No	If the ApsaraMQ for RocketMQ instance does not have separate namespaces, you do not need to configure this parameter. If the ApsaraMQ for RocketMQ instance has separate namespaces, you must configure this parameter.

Data type mapping

Data type of ApsaraMQ for RocketMQ	**Data type of Realtime Compute for Apache Flink**
STRING	VARCHAR

5.6.2.6. Create a Message Queue for Apache Kafka source table

This topic describes how to create a Message Queue for Apache Kafka source table in Realtime Compute for Apache Flink. It also describes the mappings between the values of the type parameter and Kafka versions, and provides examples on how to parse messages of Message Queue for Apache Kafka.

Important

- This topic applies only to Blink 2.0 and later.
- This topic applies only to Realtime Compute for Apache Flink that is deployed in exclusive mode.
- You can use Realtime Compute for Apache Flink to read data from a source table of a self-managed Kafka cluster. Before data is read, you must take note of the mappings between the values of the type parameter and Kafka versions, and the network configurations of the self-managed Kafka cluster and your Realtime Compute for Apache Flink cluster.
- You cannot perform local debugging on binary data. If the binary data passes the syntax check, you can perform online debugging on the binary data. For more information, see Online debugging.

Introduction to Message Queue for Apache Kafka source tables

Message Queue for Apache Kafka is a distributed, high-throughput, and scalable message queue service provided by Alibaba Cloud. Message Queue for Apache Kafka is widely used in big data scenarios, such as log collection, monitoring data aggregation, streaming data processing, and online and offline analysis. Realtime Compute for Apache Flink can use Message Queue for Apache Kafka tables as source tables or result tables to process streaming data.

The output data of Message Queue for Apache Kafka is of the serialized VARBINARY type. For each data record obtained from a Message Queue for Apache Kafka source table, you must write a user-defined table-valued function (UDTF) to parse the data into a data structure before serialization. Realtime Compute for Apache Flink first extracts data from a Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to a sink. Flink SQL also allows you to use the CAST function to parse data of the VARBINARY type into data of the VARCHAR type. For more information about UDTFs, see UDTF.

DDL syntax

The DDL definition of a Message Queue for Apache Kafka source table must be the same as the DDL definition in the following SQL statement. You can modify the settings of the parameters in the WITH clause.

```
create table kafka_stream(   --The sequence and data types of the following fields must
be the same as the five fields in the Message Queue for Apache Kafka source table.
  messageKey VARBINARY,
  `message`    VARBINARY,
  topic      VARCHAR,
  `partition`  INT,
  `offset`     BIGINT
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...
);
```

Parameters in the WITH clause

- General configurations

Parameter	Description	Required	Remarks
type	The Kafka version.	Yes	Valid values: Kafka08, Kafka09, Kafka010, and Kafka011. For more information about the mappings between the values of the type parameter and Kafka versions, see Mappings between the values of the type parameter and Kafka versions.
topic	The name of the topic to read.	Yes	N/A.
topicPattern	The regular expression used to read multiple topics.	No	Topics are separated by vertical bars (|), such as `topic1|topic2|topic3`. For more information, see Class Pattern.

startupMode	The start offset for reading data.	No **Note** We recommend that you set this parameter based on your business requirements.	Valid values: ◦ **GROUP_OFFSETS**: reads data by group. This is the default value. ◦ **EARLIEST**: reads data from the earliest offset in the Kafka cluster. ◦ **LATEST**: reads data from the latest offset in the Kafka cluster. ◦ **TIMESTAMP**: reads data from a specific point in time. You can specify the **startTimeMs** or **startTime** parameter in the WITH clause. The value of the startTimeMs parameter is a timestamp, whereas the value of the startTime parameter is in the `yyyy-MM-dd HH:mm:ss` format. We recommend that you use the **startTimeMs** parameter. If neither of the parameters is specified, data is consumed from the start offset specified by Realtime Compute for Apache Flink. **Note** ◦ If you do not specify this parameter in plaintext, GROUP_OFFSETS is used by default. ◦ If you set this parameter to TIMESTAMP, you must specify the time zone in job parameters in plaintext, such as `blink.job.timeZone=Asia/Shanghai`. ◦ In GROUP_OFFSETS mode, if no offset is configured, data is read from the end of the Kafka partition when a group consumes Kafka data for the first time. ◦ Only Kafka010 and Kafka011 support the TIMESTAMP data type.
partitionDiscoveryIntervalMS	The interval at which new partitions are checked.	No	◦ Kafka08: By default, new partitions are checked at a specific interval. ◦ Kafka09 or later: The **partitionDiscoveryIntervalMS** parameter is not supported. You can specify **extraConfig='flink.partition-discovery.interval-millis=60000'** in the WITH clause to achieve the same effect as the partitionDiscoveryIntervalMS parameter. Default value: 60000. Unit: milliseconds.

extraConfig	Additional KafkaConsumer configuration items.	No	You can use this parameter to add configuration items that are required in special scenarios but are not included in the optional configuration items. Example: `'fetch.message.max.bytes=104857600'`. Separate multiple configuration items with semicolons (;).

- Configurations for Kafka08
  - Required configurations

Parameter	Description	Required
group.id	The ID of the consumer group.	Yes
zookeeper.connect	The ZooKeeper URL.	Yes

- Optional configurations
  - consumer.id
  - socket.timeout.ms
  - fetch.message.max.bytes
  - num.consumer.fetchers
  - auto.commit.enable
  - auto.commit.interval.ms
  - queued.max.message.chunks
  - rebalance.max.retries
  - fetch.min.bytes
  - fetch.wait.max.ms
  - rebalance.backoff.ms
  - refresh.leader.backoff.ms
  - auto.offset.reset
  - consumer.timeout.ms
  - exclude.internal.topics
  - partition.assignment.strategy
  - client.id
  - zookeeper.session.timeout.ms
  - zookeeper.connection.timeout.ms
  - zookeeper.sync.time.ms
  - offsets.storage
  - offsets.channel.backoff.ms
  - offsets.channel.socket.timeout.ms
  - offsets.commit.max.retries
  - dual.commit.enabled
  - partition.assignment.strategy
  - socket.receive.buffer.bytes
  - fetch.min.bytes
- Configurations for Kafka09, Kafka010, and Kafka011
  - Required configurations

Parameter	Description
group.id	The ID of the consumer group.
bootstrap.servers	The endpoint of the Message Queue for Apache Kafka cluster.

- For more information about the optional configurations for Kafka09, Kafka010, and Kafka011, see the following Kafka documentation:
  - Kafka09
  - Kafka010
  - Kafka011

  If you want to modify the configurations, you can add parameters to the WITH clause in the DDL statement. For example, if you want to configure Simple Authentication and Security Layer (SASL), add the `security.protocol`, `sasl.mechanism`, and `sasl.jaas.config` parameters.

```
create table kafka_stream(
  messageKey varbinary,
  `message` varbinary,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...,
  `security.protocol`='SASL_PLAINTEXT',
  `sasl.mechanism`='PLAIN',
  `sasl.jaas.config`='org.apache.kafka.common.security.plain.PlainLoginModule required username="<yourUserName>" password="<yourPassword>";'
);
```

Mappings between the values of the type parameter and Kafka versions

type	Kafka version
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2 and later

Examples on how to parse messages of Message Queue for Apache Kafka

- Scenario 1: Realtime Compute for Apache Flink processes the data read from Message Queue for Apache Kafka and exports the result data to ApsaraDB RDS.

  Data stored in Message Queue for Apache Kafka is in the JSON format and must be processed by using Realtime Compute for Apache Flink. The following code shows the message format:

```
{
  "name":"Alice",
  "age":13,
  "grade":"A"
}
```

- Data processing method 1: Realtime Compute for Apache Flink reads and processes data from the Message Queue for Apache Kafka source table and then exports the result data to ApsaraDB RDS.

  In Blink 2.2.7 and later, you can use the CAST function to convert the VARBINARY data type into the VARCHAR data type. Then, use the JSON_VALUE function to parse the data of the Message Queue for Apache Kafka source table. The following code shows an example:

```
CREATE TABLE kafka_src (
  messageKey  VARBINARY,
  `message`   VARBINARY,
  topic       VARCHAR,
  `partition` INT,
  `offset`    BIGINT
) WITH (
  type = 'kafka010', -- For more information, see Mappings between the values of th
e type parameter and Kafka versions.
);

CREATE TABLE rds_sink (
  `name`       VARCHAR,
  age          VARCHAR,
  grade        VARCHAR
) WITH(
  type='rds'
);

CREATE VIEW input_view AS
  SELECT CAST(`message` as VARCHAR ) as `message`
FROM kafka_src;

INSERT INTO rds_sink
SELECT
  JSON_VALUE(`message`,'$.name'),
  JSON_VALUE(`message`,'$.age'),
  JSON_VALUE(`message`,'$.grade')
FROM input_view;
```

- Data processing method 2: Realtime Compute for Apache Flink extracts data from the Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to ApsaraDB RDS.

  To parse irregular data or complex JSON data, you must write UDTF code. Examples:

  - SQL

```
-- Define a UDTF to parse messages of Message Queue for Apache Kafka.
CREATE FUNCTION kafkaparser AS 'com.alibaba.kafkaUDTF';

-- Define a Message Queue for Apache Kafka source table. Note that the fields dec
```

```
lared in the DDL statement of the Message Queue for Apache Kafka source table mus
t be the same as the fields in the following example. You can modify the
settings of the parameters in the WITH clause.
CREATE TABLE kafka_src (
  messageKey  VARBINARY,
  `message`   VARBINARY,
  topic       VARCHAR,
  `partition` INT,
  `offset`    BIGINT
) WITH (
  type = 'kafka010', -- For more information, see Mappings between the values of
the type parameter and Kafka versions.
  topic = 'test_kafka_topic',
  `group.id` = 'test_kafka_consumer_group',
  bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);

CREATE TABLE rds_sink (
  name       VARCHAR,
  age        INT,
  grade      VARCHAR,
  updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourDatabasePassword>'
);

-- Use a UDTF to parse data of the VARBINARY type into formatted data.
CREATE VIEW input_view (
  name,
  age,
  grade,
  updateTime
) AS
SELECT
  T.name,
  T.age,
  T.grade,
  T.updateTime
FROM
  kafka_src as S,
  LATERAL TABLE (kafkaparser (`message`)) as T (
  name,
  age,
  grade,
  updateTime
);

-- Compute the formatted data and export the result data to ApsaraDB RDS.
INSERT INTO rds_sink
SELECT
```

```
  name,
  age,
  grade,
  updateTime
FROM input_view;
```

- UDTF

  For more information about how to create a UDTF, see UDTF. The following example shows Maven dependencies of Blink 2.2.4.

```
    <dependencies>
        <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-streaming-java_2.11</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>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.9</version>
        </dependency>
    </dependencies>
```

```
package com.alibaba;

import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
import java.sql.Timestamp;

public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
            /* input message :
                {
                  "name":"Alice",
                  "age":13,
                  "grade":"A",
                  "updateTime":1544173862
```

```
                    "updateTime":1544173862
                }
            */
            String msg = new String(message, "UTF-8");
            try {
                JSONObject data = JSON.parseObject(msg);
                if (data != null) {
                    String name = data.getString("name") == null ? "null" : data.g
etString("name");
                    Integer age = data.getInteger("age") == null ? 0 : data.getInt
eger("age");
                    String grade = data.getString("grade") == null ? "null" : data
.getString("grade");
                    Timestamp updateTime = data.getTimestamp("updateTime");

                    Row row = new Row(4);
                    row.setField(0, name);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3,updateTime );

                    System.out.println("Kafka message str ==>" + row.toString());
                    collect(row);
                }
            } catch (ClassCastException e) {
                System.out.println("Input data format error. Input data " + msg +
"is not json string");
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
    @Override
    // If the return value is declared as a row, you must reload the UDTF method
and specify the types of fields to be returned.
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.STRING, DataTypes.INT,
DataTypes.STRING, DataTypes.TIMESTAMP);
    }
}
```

- Scenario 2: Realtime Compute for Apache Flink reads data from a Message Queue for Apache Kafka source table and processes the data by using window functions.

You must define watermarks in the DDL statement of a source table for windows, such as tumbling and sliding windows, based on the design of Realtime Compute for Apache Flink. For more information, see Watermark. The method you use to define a watermark in a Message Queue for Apache Kafka source table is different from the method you use for other types of source tables. If you want to perform an event time-based computation by using a window function, you must use a user-defined extension (UDX) to parse the event time in the message field of a source table. Then, you can define a watermark based on the parsed event time. You must use a computed column to convert data types for the event time parsed from a Message Queue for Apache Kafka source table. For example, the data `2018-11-11 00:00:00|1|Anna|female` is written to the Message Queue for Apache Kafka source table. During the computing process, Realtime Compute for Apache Flink extracts data from the Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to ApsaraDB RDS.

- Data processing method 1: Realtime Compute for Apache Flink reads and processes data from the Message Queue for Apache Kafka source table and then exports the result data to ApsaraDB RDS.

  In Blink 2.2.7 and later, you can use the CAST function to convert the VARBINARY data type into the VARCHAR data type. Then, use the JSON_VALUE function to parse the data of the Message Queue for Apache Kafka source table. The following code shows an example:

```
CREATE TABLE kafka_src (
  messageKey VARBINARY,
  `message` VARBINARY,
  topic VARCHAR,
  `partition` INT,
  `offset` BIGINT,
  ts as to_timestamp(json_value(cast(`message` as VARCHAR ),'$.nodes.time')),
  WATERMARK wk FOR ts as withOffset(ts, 2000)
) WITH (type = 'kafka' -- For more information, see Mappings between the values of
the type parameter and Kafka versions.
);

CREATE TABLE rds_sink (
  starttime TIMESTAMP ,
  endtime   TIMESTAMP ,
  `message` BIGINT
) WITH (type = 'rds');

INSERT
  INTO rds_sink
SELECT
  TUMBLE_START(ts, INTERVAL '1' MINUTE),
  TUMBLE_END(ts, INTERVAL '1' MINUTE),
  count(`message`)
FROM
  kafka_src
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE);
```

- Data processing method 2: Realtime Compute for Apache Flink extracts data from the Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to ApsaraDB RDS.
  - SQL

```
-- Define a UDTF to parse messages of Message Queue for Apache Kafka.
```

```
CREATE FUNCTION kafkapaser AS 'com.alibaba.kafkaUDTF';
CREATE FUNCTION kafkaUDF AS 'com.alibaba.kafkaUDF';

-- Define a Message Queue for Apache Kafka source table. Note that the fields def
ined in the DDL statement must be the same as the fields in the following stateme
nt. You can modify the settings of the parameters in the WITH clause.
create table kafka_src (
  messageKey VARBINARY,
  `message` VARBINARY,
  topic VARCHAR,
  `partition` INT,
  `offset` BIGINT,
  ctime AS TO_TIMESTAMP(kafkaUDF(`message`)), -- Define a computed column. A comp
uted column can be considered as a placeholder column that is not stored in a sou
rce table. The values in this column are computed. If you want to define a waterm
ark, the data type of the computed column must be TIMESTAMP.
  watermark for `ctime` as withoffset(`ctime`,0) -- Define a watermark in a compu
ted column.
) WITH (
  type = 'kafka010', -- For more information, see Mappings between the values of
the type parameter and Kafka versions.
  topic = 'test_kafka_topic',
  `group.id` = 'test_kafka_consumer_group',
  bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);

create table rds_sink (
  `name` VARCHAR,
  age INT,
  grade VARCHAR,
  updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourPassword>'
);


-- Use a UDTF to parse data of the VARBINARY type to formatted data.
CREATE VIEW input_view AS
SELECT
  S.ctime,
  T.`order`,
  T.`name`,
  T.sex
  from
  kafka_src as S,
  LATERAL TABLE (kafkapaser (`message`)) as T (
  ctime,
  `order`,
  `name`,
  sex
);
```

```
);

-- Compute the data in input_view.
CREATE VIEW view2 (
  cnt,
  sex
) AS
  SELECT
  COUNT(*) as cnt,
  T.sex
  from
  input_view
Group BY sex, TUMBLE(ctime, INTERVAL '1' MINUTE);

-- Compute the formatted data and export the result data to ApsaraDB RDS.
insert into rds_sink
  SELECT
  cnt,sex
from view2;
```

- UDF&UDTF

  For more information about how to create UDFs and UDTFs, see UDF and UDTF. The following example shows Maven dependencies of Blink 2.2.4.

```
  <dependencies>
        <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-streaming-java_2.11</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>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.9</version>
        </dependency>
    </dependencies>
```

- UDTF

```
package com.alibaba;

import com.alibaba.fastjson.JSONObject;
```

```
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;

/**
  The following example shows how to parse the JSON strings in a Message Queue
for Apache Kafka source table and format the parsed data.
**/
public class kafkaUDTF extends TableFunction<Row> {

    public void eval(byte[] message) {
        try {
          // Read data of the VARBINARY data type and convert the data into the
STRING data type.
            String msg = new String(message, "UTF-8");

                // Extract data from JSON strings based on the following fields:

                    String ctime = Timestamp.valueOf(data.split('\\|')[0]);
                    String order = data.split('\\|')[1];
                    String name = data.split('\\|')[2];
                    String sex = data.split('\\|')[3];

                    // Return rows of data based on the parsed fields.
                    Row row = new Row(4);
                    row.setField(0, ctime);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3, updateTime);

                    System.out.println("Kafka message str ==>" + row.toString())
;

                    // Return a row of data.
                    collect(row);

            } catch (ClassCastException e) {
                System.out.println("Input data format error. Input data " + msg
+ "is not json string");
            }


        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

    }

    @Override
    // If the return value is declared as a row, you must reload the UDTF metho
d and specify the types of fields to be returned.
    // Define the data types for objects in output rows.
```

```
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.TIMESTAMP,DataTypes.STRING, DataTypes.Integer, DataTypes.STRING,DataTypes.STRING);
    }

}
```

- UDF

```
package com.alibaba;
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;

public class KafkaUDF extends ScalarFunction {
    // The open method is optional.
    // To implement the open method, you must add "import org.apache.flink.table.functions.FunctionContext;" to the code.

    public String eval(byte[] message) {

         // Read data of the VARBINARY data type and convert it into the STRING data type.
        String msg = new String(message, "UTF-8");
        return msg.split('\\|')[0];
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    // The close method is optional.
    @Override
    public void close() {
        }
}
```

Create a source table of a self-managed Kafka database

- Example

```
create table kafka_stream(
  messageKey VARBINARY,
  `message` VARBINARY,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka011',
  topic = 'kafka_01',
  `group.id` = 'CID_blink',
  bootstrap.servers = '192.168.0.251:****'
);
```

- Parameters in the WITH clause

  For more information, see Parameters in the WITH clause.

> **Note**
>
> - **bootstrap.servers** specifies the address and port number of the self-managed Kafka cluster.
> - Metrics, such as transactions per second (TPS) and requests per second (RPS) of Message Queue for Apache Kafka or a self-managed Kafka database, can be displayed only in the console of Realtime Compute for Apache Flink of Blink 2.2.6 and later.

FAQ

- Q: What do I do if the following error occurs when a job is being started?

```
ERR_ID:
     SQL-00010007
CAUSE:
     Could not create table 'kafka_source' as source table
ACTION:
     Please refer to details section for hint.
     If it doesn't help, please contact customer support
DETAIL:
     java.lang.IllegalArgumentException: Startup time[1566481803000] must be before c
urrent time[1566453003356].
```

  A: This error is caused by invalid time zone settings. To solve this issue, add the following parameter to job parameters:

```
blink.job.timeZone=Asia/Shanghai
```

- Q: What do I do if a self-managed Kafka cluster cannot consume data?

  A:

  - Causes

    Each broker of the Kafka cluster sends its metadata to ZooKeeper. Then, the Kafka consumer accesses a broker to extract data by using the endpoint in listener_security_protocol_map in the metadata of the broker. The endpoint is either the IP address or the combination of the local server domain name and port number. If the machine where your Realtime Compute for Apache Flink job resides cannot access the **endpoint**, the **consumer** of the **connector** cannot extract data. As a result, the data consumption process stops.

  - Troubleshooting

    a. View the endpoint in listener_security_protocol_map of the broker of ZooKeeper.

    b. Use the **network detection** feature to check whether you can access the IP address or domain name in the **endpoint**.

    c. Log on to your machine to confirm the cause.

- Solution
  - If the **endpoint** is an IP address, check whether a whitelist for accessing storage resources is configured for the Kafka server. If no whitelist is configured, configure a whitelist and try again.
  - Exclusive clusters of Realtime Compute for Apache Flink cannot resolve domain names. If the **endpoint** is a domain name and a whitelist has been configured, use one of the following methods:
    - If you cannot restart the Kafka service, perform the following operations:

      Purchase PrivateZone and configure domain name resolution for all Kafka brokers. After network detection based on the domain name succeeds, restart your Realtime Compute for Apache Flink job.
    - If you can restart the Kafka service, perform the following operations:

      Configure the IP address and port number (in **boostrap.servers** in typical cases) for **advertised.listeners** for the related broker. Make sure that the network connection is normal. Then, restart your Realtime Compute for Apache Flink job.

- Q: Why does a job terminate immediately after it is started during the consumption of Kafka data?

  A: For versions earlier than Blink 3.3.0, if the startupMode parameter is set to **TIMESTAMP** and all the partitions of Kafka contain no data, the Kafka connector determines that no partition data can be consumed and terminates the job. You can view log information similar to `Consumer subtask {} initially has no partitions from which to read.` in the `TaskManager.log` log filethat corresponds to the job. We recommend that you upgrade the Blink version to 3.3.0 or later.

- Q: What are the features of the commit offset mechanism in Realtime Compute for Apache Flink?

  A: By default, Realtime Compute for Apache Flink uses **commitOffsetOnCheckpointing**. The commit offset policy configured by users does not take effect. If you enable **checkpointing**, Realtime Compute for Apache Flink **commits** the **offset** that is consumed at the current time to Kafka each time a checkpoint is generated. This way, data is consumed from the offset that is **committed** from the last **checkpoint** during job restoration. This ensures exactly-once processing of streaming data. If the **checkpoint** interval exceeds the specified upper limit, Kafka may fail to query the consumed **offset**.

5.6.2.7. Create a Tablestore source table

This topic describes how to create a Tablestore source table in Realtime Compute for Apache Flink.

> **Important** This topic applies only to Realtime Compute for Apache Flink V3.2.2 and later.

Introduction to Tablestore

Tablestore is a distributed NoSQL database service that is built on the Apsara distributed operating system of Alibaba Cloud. Tablestore adopts data sharding and load balancing technologies to scale out and handle concurrent transactions. You can use Tablestore to store and query a large amount of structured data in real time.

Tunnel Service of Tablestore

Tunnel Service is a centralized service that uses the Tablestore API to allow you to consume full and incremental data. You can use the Tunnel Service API and SDKs to create tunnels from which you can consume distributed data in real time. The distributed data is divided into

the following types: incremental data, full data, and full and incremental data. You can use the tunnels of the Tunnel Service to consume existing data or added data in tables in streaming processing mode. In Realtime Compute for Apache Flink, the tunnels of the Tunnel Service can serve as the source of streaming data. Each data record uses a JSON-like format. You can run the following code if you need to use Tunnel Service tunnels:

```
{
            "OtsRecordType": "PUT",  // The data operation type, such as PUT, UPDATE,
and DELETE.
            "OtsRecordTimestamp": 1506416585740836, // The time when the data is
written. The time unit is milliseconds. The value 0 indicates that full data is written
.
            "PrimaryKey": [
                {
                    "ColumnName": "pk_1", // The first primary key column.
                    "Value": 1506416585881590900
                },
                {
                    "ColumnName": "pk_2", // The second primary key column.
                    "Value": "string_pk_value"
                }
            ],
            "Columns": [
                {
                    "OtsColumnType": "Put", // The operation type for the column, such a
s PUT, DELETE_ONE_VERSION, and DELETE_ALL_VERSION.
                    "ColumnName": "attr_0",
                    "Value": "hello_table_store",
                },
                {
                    "OtsColumnType": "DELETE_ONE_VERSION", // No value is specified for
the delete operation.
                    "ColumnName": "attr_1"
                }
            ]
}
```

DDL syntax

In Realtime Compute for Apache Flink, you can use Tablestore to store input data. The following code shows an example:

```
create table tablestore_stream(
  pk_1 BIGINT,
  pk_2 VARCHAR,
  attr_0 VARCHAR,
  attr_1 DOUBLE,
  OtsRecordType  VARCHAR  HEADER //You must add HEADER to the attribute field.
) with (
  type ='ots',
  endPoint ='http://blink-demo.cn-hangzhou.vpc.tablestore.aliyuncs.com',
  instanceName = 'blink-demo',
  tableName ='demo_table',
  tunnelName = 'blink-demo-stream',
  accessId ='<yourAccessID>',
  accessKey ='<yourAccessSecret>',
  ignoreDelete = 'false' // Specifies whether to ignore the delete operation.
);
```

Attribute fields

For more information about how to obtain and use attribute fields in a Tablestore source table, see Obtain attribute fields of a source table.

Field	Description
`OtsRecordType`	The data operation type.
`OtsRecordTimestamp`	The data operation time. If you set this parameter to 0, full data is written.
`<Column name>_OtsColumnType`	The operation type for a column.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the connector.	Set the value to `ots`.
endPoint	The endpoint of the Tablestore instance.	Enter the VPC endpoint if the instance is deployed in a VPC.
instanceName	The name of the Tablestore instance.	None.
tableName	The name of the Tablestore table.	Realtime Compute for Apache Flink does not reread data that has been read from a Tablestore source table. If you want Realtime Compute for Apache Flink to reread full data from the source table, you must create a data tunnel.
tunnelName	The tunnel name of the Tablestore source table.	None.

accessId	The AccessKey ID that is used to access Tablestore.	None.
accessKey	The AccessKey secret that is used to access Tablestore.	None.
ignoreDelete	Specifies whether to ignore the delete operation.	Optional. Default value: `false`.

5.6.2.8. Create a full MaxCompute source table

This topic describes how to create a full MaxCompute source table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause and data type mappings used when you create a full MaxCompute source table.

> **Important**
>
> - This topic applies only to Blink 2.2.7 and later.
> - Full MaxCompute source tables are typically used as bounded stream tables. This makes MaxCompute different from other data sources, such as DataHub and Kafka. In Blink 3.4.4, you can specify a full MaxCompute source table as an unbounded stream table. This way, the source table can continuously listen to new partitions. If a new partition is generated, Realtime Compute for Apache Flink reads data from the new partition. This feature is deprecated in Blink 3.5.0. To use a MaxCompute source table as an unbounded stream table, create an incremental MaxCompute source table. For more information, see Create an incremental MaxCompute source table.

DDL syntax

In Realtime Compute for Apache Flink, you can use MaxCompute to store input data. The following code shows an example:

```
create table odps_source(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  type = 'odps',
  endPoint = 'http://service.cn.maxcompute.aliyun-inc.com/api',
  project = '<projectName>',
  tableName = '<tableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****' --If your MaxCompute source table is a non-partitioned table, you do not need to declare this parameter.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks

endPoint	The endpoint of MaxCompute.	Yes	For more information, see Endpoints.
tunnelEndpoint	The endpoint of MaxCompute Tunnel.	No	For more information, see Endpoints. **Note** This parameter is required if MaxCompute is deployed in a virtual private cloud (VPC).
project	The name of the MaxCompute project.	Yes	N/A.
tableName	The name of the MaxCompute table.	Yes	N/A.
accessId	The AccessKey ID that is used to access MaxCompute.	Yes	N/A.
accessKey	The AccessKey secret that is used to access MaxCompute.	Yes	N/A.

partition	The name of a partition.	No	• A full MaxCompute table that has only one-level partitions For example, if only one partition key column `ds` exists, `` `partition` = 'ds=20180905' `` indicates that data in the `ds=20180905` partition is read. • A full MaxCompute table that has multi-level partitions For example, if two partition key columns `ds` and `hh` exist, `` `partition`='ds=20180905,hh=*' `` indicates that data in the `ds=20180905` partition is read. **Note** When you filter partitions, you must declare the values of all partitions. In the preceding example, if you declare only `` `partition` = 'ds=20180905' ``, no partition data is read.
subscribeNewPartition	Specifies whether to listen to new partitions that meet specific conditions.	No	Default value: false. This value indicates that the system does not listen to new partitions. **Note** • If the **subscribeNewPartition** parameter is set to `true`, you cannot specify the value of the partition parameter. Otherwise, new partitions cannot be read. • This parameter is provided only in Blink 3.4.4. The parameter is deprecated in Blink 3.5.0. If you need to use this parameter, create an incremental MaxCompute source table. For more information, see Create an incremental MaxCompute source table.

subscribeIntervalInSec	The interval at which new partitions are listened to.	No	Default value: 30. Unit: seconds. **Note** If the value of this parameter is too small, pressure may be caused on the MaxCompute metadata service. This may result in failures to listen to the service.
maxPartitionCount	The number of partitions in the partitioned table that is read if the partition parameter is not specified.	No	Default value: 100. **Note** Only Blink 3.0 and later support this parameter.

Data type mappings

Data type of MaxCompute	Data type of Realtime Compute for Apache Flink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
DECIMAL	DECIMAL

BINARY	VARBINARY
STRING	VARCHAR

Sample code

The following sample code shows how to create a full MaxCompute source table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE odps_source (
  cid varchar,
  rt DOUBLE,
) with (
  type = 'odps',
  endPoint = '<yourEndpointName>',
  project = '<yourProjectName>',
  tableName = '<yourTableName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessPassword>',
  partition = 'ds=20190712'
);

CREATE TABLE test (
  cid varchar,
  invoke_count BIGINT
) with (
  type='print'
);

INSERT INTO test
SELECT
  cid,
  count(*) as invoke_count
FROM odps_source GROUP BY cid;
```

FAQ

- Q: What do I do if the values of the endPoint and tunnelEndpont parameters in the DDL statement are incorrect?

  A: For more information about the endPoint and tunnelEndpont parameters, see Endpoints. Incorrect configuration of parameters may lead to the following issues:

  - If the configuration of the endPoint parameter is incorrect, the task publish progress stops at 91%.
  - If the tunnelEndpoint parameter is incorrectly configured, the task fails.

- Q: How does the full MaxCompute data storage read data in a full MaxCompute source table?

  A: The full MaxCompute data storage reads data from the full MaxCompute source table by using a tunnel. Therefore, the read speed and bandwidth are restricted by the bandwidth of the tunnel used by the full MaxCompute source table.

- Q: If data of some partitions of a full MaxCompute source table has been read, can the full MaxCompute data storage read data that is newly written to these partitions after a Realtime Compute for Apache Flink job is started?

  A: No, the full MaxCompute data storage cannot read the new data from the partitions. The full MaxCompute data storage reads data from tables or partitions by using a tunnel. After a Realtime Compute for Apache Flink job is started, MaxCompute Reader exits when data reading is complete. Then, MaxCompute Reader does not read new data from the full MaxCompute source table or partitions.

- Q: How does the full MaxCompute data storage read data that is newly written to the full MaxCompute source table or partitions after a Realtime Compute for Apache Flink job is started?

  A: Realtime Compute for Apache Flink V3.4 and later support the **subscribeNewPartition** parameter that determines whether to listen to new partitions. New data can be written to new partitions. The following code shows an example:

```
CREATE TABLE blink_source (
    cid varchar,
    rt DOUBLE,
) with (
    type = 'odps',
    endPoint = '<yourEndpointName>',
    project = '<yourProjectName>',
    tableName = '<yourTableName>',_table_name',
    subscribeNewPartition = 'true'
    -- You cannot specify the partition parameter if you want to listen to new partit
ions.
    accessId = '<yourAccessKeyId>',
    accessKey = '<yourAccessKeySecret>',
);

CREATE TABLE test (
    cid varchar,
    invoke_count BIGINT
) with (
  type='print'
);

INSERT INTO test
SELECT
    cid,
    count(*) as invoke_count
FROM blink_source GROUP BY cid;
```

  > **Note**
  >
  > Data that is generated for new partitions in a full MaxCompute source table must be written to the new partitions of the table in Realtime Compute for Apache Flink. The data that is written to existing partitions is invalid.

- Q: Can I use `max_pt()` or `max_pt_with_done()` in the value of the **partition** parameter in the WITH clause?

A: We recommend that you do not use these parameters in the WITH clause. If you want to use these parameters, make sure that you understand the usage of `max_pt()` in a full MaxCompute source table in the following scenarios:

- Listening to new partitions is not enabled.

  After a task is started, MaxCompute Reader uses the full MaxCompute metadata service to obtain all partitions in the current full MaxCompute source table and reads `max_pt()`. After data reading is complete, MaxCompute Reader exits and does not read new data from the partition to which max_pt belongs or listen to new partitions.

- Listening to new partitions is enabled.

  After a task is started, MaxCompute Reader uses the full MaxCompute metadata service to obtain all partitions in the current full MaxCompute source table and reads `max_pt()`. After data reading is complete, MaxCompute Reader does not read new data from the partition to which max_pt() belongs. However, MaxCompute Reader listens to the generation of new partitions at specific intervals. For more information, see **subscribeIntervalInSec**. If new partitions are generated, MaxCompute Reader reads the partitions and then reads `max_pt()` from the partitions. After data reading is complete, the system waits for the next listening event. If no new partitions are generated, the system waits for the next listening event.

- Q: If a full MaxCompute source table is referenced as a data source, can the data that is appended to an existing partition or table be read after a job is started?

  A: No, the data cannot be read and the job may fail. The full MaxCompute data storage uses `ODPS DOWNLOAD SESSION` to read data from tables or partitions. When you create a `DOWNLOAD SESSION`, the MaxCompute server creates an index file, which contains the data mapping obtained when the `DOWNLOAD SESSION` is created. Subsequent data reading is performed based on the data mapping. Therefore, the data that is appended to the full MaxCompute source table or partitions after the `DOWNLOAD SESSION` is created cannot be read in normal cases. This issue occurs in the following scenarios:

  - When the MaxCompute data storage reads data by using a tunnel, the following error is returned if new data is written to the table or partitions in the table: `ErrorCode=TableModified,ErrorMessage=The specified table has been modified since the download initiated.`
  - New data is written to the table or partitions in the table. However, the tunnel through which data is read is disabled. Therefore, the new data cannot be read. If a job is recovered from failure or is resumed, the data may be incorrect. For example, existing data is read again but the newly added data may not be read completely.

- Q: Can I suspend and resume a job for a full MaxCompute source table? Can I change the parallelism of the full MaxCompute source table?

  A: No, you cannot suspend or resume a job for a full MaxCompute source table or change the parallelism of the full MaxCompute source table. MaxCompute determines which data in which partitions need to be read for each parallel job and records the consumption information for each parallel job in the state based on the parallelism. This way, MaxCompute can continue reading data from the last read position after the job is suspended and then resumed or fails. This logic is based on the premise that the parallelism is configured. If you suspend and then resume a job for a full MaxCompute source table after you change the parallelism of the source table, the impact on the job cannot be estimated because some data may be repeatedly read but some data may not be read.

- Q: Why are the partitions before the start offset also read when you set the start time to `2019-10-11 00:00:00` for a job?

A: The start time is valid only for data sources of the message queue type, such as DataHub. The start time is invalid for full MaxCompute source tables. After you start a Realtime Compute for Apache Flink job, Realtime Compute for Apache Flink reads data in the following ways:

- For a partitioned table, fully managed Flink reads data from all existing partitions.
- For a non-partitioned table, fully managed Flink reads all existing data.

- Q: What do I do if the error message "ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'" appears when a job is running?

  A: This error occurs because the user identity information specified in the MaxCompute DDL statements cannot be used to access MaxCompute. Therefore, you must use an Alibaba Cloud account, a RAM user, or a RAM role to authenticate the user identity. For more information, see User authentication.

5.6.2.9. Create an incremental MaxCompute source table

This topic describes how to create an incremental MaxCompute source table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, data type mappings, and FAQ involved when you create an incremental MaxCompute source table.

> **Important**
> - This topic applies only to Blink 3.5.0-hotfix and later.
> - Incremental MaxCompute source tables cannot be used as dimension tables.
> - Incremental MaxCompute source tables must be partitioned tables.

DDL syntax

```
create table odps_source(
    id int,
    user_name VARCHAR,
    content VARCHAR
) with (
    type = 'continuous-odps',
    endPoint = 'your_end_point_name',
    project = 'your_project_name',
    tableName = 'your_table_name',
    accessId = 'your_access_id',
    accessKey = 'your_access_key',
    startPartition = 'ds=20180905'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks

type	The type of the connector.	Yes	Set the value to `continuous-odps`.
endPoint	The endpoint of MaxCompute.	Yes	For more information, see Endpoints.
tunnelEndpoint	The endpoint of the MaxCompute Tunnel service.	• Yes: This parameter is required if MaxCompute is deployed in a virtual private cloud (VPC). • No: This parameter is not required if MaxCompute is not deployed in a VPC.	For more information, see Endpoints.
project	The name of the project to which the table belongs.	Yes	N/A.
tableName	The name of the table.	Yes	N/A.
accessId	The AccessKey ID that is used to access MaxCompute.	Yes	N/A.
accessKey	The AccessKey secret that is used to access MaxCompute.	Yes	N/A.

startPartition	The start partition from which data is read. When the system loads partitioned tables, the system compares **startPartition** with all partitions in each partitioned table in alphabetical order and then loads the data that meets the specified condition from the partitions. An incremental MaxCompute source table can also continuously listen to incremental MaxCompute partitioned tables. After the source table reads data from the existing partitions, the source table continues to listen to the generation of new partitions. After a new partition is generated, the source table reads data from the new partition. **Note** • An incremental MaxCompute source table must have first-level partitions. Second-level partitions are optional. • If you specify a second-level partition, make sure that the second-level partition is placed after a first-level partition. • If the partition specified by the **startPartition** parameter does not exist, the next partition is used as the start partition.	Yes	For example, if startPartition is set to **ds=20191201**, data of all the partitions that meets the condition of **ds >= 20191201** in the incremental MaxCompute partitioned table is loaded. For example, an incremental MaxCompute partitioned table has the first-level partition key column ds and the second-level partition key column type and contains the following partitions: • ds=20191201,type=a • ds=20191201,type=b • ds=20191202,type=a • ds=20191202,type=b • ds=20191202,type=c The partitions from which data is read vary based on the setting of **startPartition**: • ds=20191202 ◦ ds=20191202,type=a ◦ ds=20191202,type=b ◦ ds=20191202,type=c • ds=20191201,type=c ◦ ds=20191202,type=a ◦ ds=20191202,type=b ◦ ds=20191202,type=c

Data type mappings

Data type of MaxCompute	Data type of Realtime Compute for Apache Flink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

Important

- Incremental MaxCompute source tables do not support the CHAR, VARCHAR, ARRAY, MAP, or STRUCT data type.
- You can use the STRING data type instead of the VARCHAR data type.

Sample code

One partition is generated in an incremental MaxCompute source table every day. The partition key column is ds. The incremental MaxCompute source table loads data from partitions whose partition names are greater than or equal to **20191201** and continuously listens to the generation of new partitions.

```
-- The incremental MaxCompute source table reads data from partitions in the range of [
ds=20191201, ∞).
CREATE TABLE odps_source (
    cid VARCHAR,
    rt DOUBLE,
) with (
    type = 'continuous-odps',
    endPoint = 'your_end_point_name',
    project = 'your_project_name',
    tableName = 'your_table_name',
    accessId = 'xxxx',
    accessKey = 'xxxx',
    startPartition = 'ds=20191201'
);

CREATE TABLE test (
    cid VARCHAR,
    rt DOUBLE,
) with (
    type='print'
);

INSERT INTO test
SELECT
    cid, rt FROM odps_source;
```

FAQ

- Q: What do I do if the values of the endPoint and tunnelEndpont parameters in the DDL statement are incorrect?

  A: For more information about the endPoint and tunnelEndpont parameters, see Endpoints in different regions (Internet). Incorrect configuration of parameters may lead to the following issues:

  - If the configuration of the endPoint parameter is incorrect, the task publish progress stops at 91%.
  - If the tunnelEndpoint parameter is incorrectly configured, the task fails.

- Q: How does the incremental MaxCompute data storage read data in an incremental MaxCompute source table?

  A: The incremental MaxCompute data storage reads MaxCompute data by using a tunnel. Therefore, the read speed and bandwidth are restricted by the bandwidth of the tunnel used by the incremental MaxCompute source table.

- Q: If data of some partitions of an incremental MaxCompute source table has been read, can the incremental MaxCompute data storage read data that is newly written to these partitions after a Realtime Compute for Apache Flink job is started?

  A: No, the incremental MaxCompute data storage cannot read the new data from the partitions. The incremental MaxCompute data storage reads data from the partitions of an incremental MaxCompute source table by using a tunnel. After a Realtime Compute for Apache Flink job is started, MaxCompute Reader exits when data reading is complete. Then, MaxCompute Reader does not read new data from the partitions of the incremental MaxCompute source table.

- Q: How do I view partition names of an incremental MaxCompute source table?

A: You can perform the following steps to view the partition names of an incremental MaxCompute source table on the table details page:

i. Go to the Data Map to search for the required table name.

ii. Click the table name.

iii. On the right side of the table details page, click the **Details** tab and then the **Partitions** tab. You can view MaxCompute partition names in the **Partition Name** column.

- Q: If an incremental MaxCompute source table is referenced as a data source, can the data that is appended to an existing partition be read after a job is started?

  A: No, the data cannot be read and the job may fail. The MaxCompute data storage uses `ODPS DOWNLOAD SESSION` to read data from tables or partitions. When you create a `DOWNLOAD SESSION`, the MaxCompute server creates an index file, which contains the data mapping obtained when the `DOWNLOAD SESSION` is created. Subsequent data reading is performed based on the data mapping. Therefore, in most cases, the data that is appended to a MaxCompute table or to a partition in the table after you create a `download session` cannot be read. This issue occurs in the following scenarios:

  - When the MaxCompute data storage reads data by using a tunnel, the following error is returned if new data is written to the table or partitions in the table: `ErrorCode=TableModified,ErrorMessage=The specified table has been modified since the download initiated.`
  - New data is written to the table or partitions in the table. However, the tunnel through which data is read is disabled. Therefore, the new data cannot be read. If a job is recovered from failure or is resumed, the data may not be correct. For example, existing data is read again but the newly added data may not be read completely.

- Q: Can I suspend and resume a job for an incremental MaxCompute source table? Can I also change the parallelism of the incremental MaxCompute source table?

  A: No, you cannot suspend or resume a job for an incremental MaxCompute source table, or change the parallelism of the incremental MaxCompute source table. MaxCompute determines which data in which partitions need to be read for each parallel job and then records the consumption information about each parallel job in state data based on the parallelism. This way, MaxCompute can continue to read data from the most recent read position after a job is suspended and then resumed or after a job is recovered from a failover.

  If you suspend and then resume a job for an incremental MaxCompute source table after you change the parallelism of the source table, the impact on the job cannot be estimated because some data may be repeatedly read and some data may not be read.

- Q: What does an incremental MaxCompute source table do if it finds that some data is not written to the new partition it detects?

  A: No mechanism is available to indicate whether data in a partition is complete. If an incremental MaxCompute source table detects a new partition, the source table immediately reads data from the partition. If the incremental MaxCompute source table reads MaxCompute partitioned table T with the partition key column of ds, data in table T is written in the following methods:

  - Not recommended: Create a partition, such as **ds=20191010** and write data to it. If the incremental MaxCompute source table consumes table T and detects the new partition **ds=20191010**, the source table immediately reads data from the new partition. If the data written to the partition is incomplete, the data fails to be read.
  - Recommended: Execute the **Insert overwrite table T partition (ds='20191010') ...** statement without the need to create a partition. After the job succeeds, both the partition and data are displayed.

- Q: What do I do if the error message "ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'" appears when a job is running?

A: This error occurs because the user identity information specified in the MaxCompute DDL statements cannot be used to access MaxCompute. Therefore, you must use an Alibaba Cloud account, a RAM user, or a RAM role to authenticate the user identity. For more information, see User authentication.

5.6.3. Create a result table

5.6.3.1. Overview of result tables

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 result table.

Data can be written into the target storage system by using one of the following methods: append and update.

- Append: If the result table is stored in Log Service, Message Queue (MQ), or an ApsaraDB for RDS database with the primary key undefined, output data is appended to the result table. The original data in the result table is not modified.
- Update: If the result table is stored in an ApsaraDB for RDS 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 does not exist in the result table, the data record is inserted into the database.
  - If the value for the primary key exists in the result table, the existing data record in the database is overwritten.

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'
);
```

5.6.3.2. Create an Oracle database result table

This topic describes how to create an Oracle database result table. It also describes the parameters in the WITH clause, data type mapping, and precautions.

> **Important**
> - This topic applies only to Blink 3.6.0 and later.
> - You can create Oracle database result tables only when you use Oracle 19c.

Precautions

- In Realtime Compute for Apache Flink, an SQL statement is executed to write each row of result data to the result table in the destination database.
- Realtime Compute for Apache Flink performs the following operations based on whether the required table exists in the Oracle database:
  - If the required table does not exist, Realtime Compute for Apache Flink creates a table in the Oracle database to store the result data.
  - If the required table exists, Realtime Compute for Apache Flink writes or updates data to the result table.
- The primary keys of logical and physical tables can be different. The primary keys of logical tables must contain the primary keys of physical tables.
- Realtime Compute for Apache Flink uses the APPEND function or the UPSERT function to insert data into an Oracle database result table.
  - If no primary key is specified in the table, the APPEND function is used.
  - If a primary key is specified in the table, the UPSERT function is used. If the primary key does not exist, the primary key is inserted. If the primary key exists, the primary key is updated.

Syntax

In Realtime Compute for Apache Flink, you can use an Oracle database to store output data. The following code shows an example:

```
CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employee_age INT,
  PRIMARY KEY(employee_id)
) WITH (
    type = 'oracle',
    url = '<yourUrl>',
    userName = '<yourUserName>',
    password = '<yourPassword>',
    tableName = '<yourTableName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to oracle.
url	The connection string of the database.	Yes	`jdbc:oracle:thin:@192.168.171.62:1521:sit0`

userName	The username that is used to log on to the database.	Yes	None.
password	The password that is used to log on to the database.	Yes	None.
tableName	The name of the table in the database.	Yes	None.
maxRetryTimes	The maximum number of retries for writing data to a table.	No	Default value: 10.
batchSize	The number of data records that are written at a time. **Note** • The **batchSize** parameter takes effect only when the primary key is defined. • Write operations are triggered if the value of the **batchSize** or **bufferSize** parameter reaches the specified threshold.	No	Default value: 50.
bufferSize	The number of data records in the buffer after duplicates are removed. **Note** • The **bufferSize** parameter takes effect only when the primary key is defined. • Write operations are triggered if the value of the **batchSize** or **bufferSize** parameter reaches the specified threshold.	No	Default value: 500.
flushIntervalMs	The write timeout period. Unit: milliseconds. **Note** If no data is written to the database within the period specified by this parameter, the system writes the cached data to the result table again.	No	Default value: 500.
excludeUpdateColumns	Specifies whether to skip the specified columns when the data of a row in the table is updated. **Note** When the data of a row in the table is updated, the primary key is not updated by default.	No	This parameter is empty by default.

ignoreDelete	Specifies whether to skip delete operations.	No	Default value: false.

Mapping between field data types

Data type of the Oracle database	Data type of Realtime Compute for Apache Flink
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

Sample code

The following sample code shows how to create an Oracle database result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE oracle_source (
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT
) WITH (
  type = 'random'
);

CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT,
  primary key(employee_id)
)with(
  type = 'oracle',
  url = 'jdbc:oracle:thin:@192.168.171.62:1521:sit0',
  userName = 'blink_test',
  password = 'blink_test',
  tableName = 'oracle_sink'
);

INSERT INTO oracle_sink
SELECT * FROM oracle_source;
```

5.6.3.3. Create a Hologres result table

This topic describes how to create a Hologres result table. It also describes the parameters in the WITH clause, streaming semantics, and data type mapping involved when you create a Hologres result table.

> **Important**
>
> - This topic applies only to Blink 3.6.0 and later. If your Blink version is earlier than 3.6.0, you can
> - Update your Blink version to 3.6.0 or later. For more information, see Manage Blink versions of a Realtime Compute for Apache Flink cluster deployed in exclusive mode.
> - Download and Install blink-connector-hologres-blink-3.6.8.jar or blink-connector-hologres-blink-3.7.jar package.
> - We recommend that you use Hologres 0.7 or later.
> - Hologres writes data asynchronously. Therefore, you must add **blink.checkpoint.fail_on_checkpoint_error=true** to the code so that a failover is triggered only when a job exception occurs.

Introduction to Hologres

Hologres is compatible with the PostgreSQL protocol and integrates seamlessly with the big data ecosystem. Hologres supports real-time analysis and the processing of petabytes of data with high concurrency and low latency. This allows you to use existing Business Intelligence (BI) tools to easily perform multidimensional analysis and business exploration.

Syntax

In Realtime Compute for Apache Flink, you can use Hologres to store output data. The following code shows an example:

```
create table Hologres_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='<yourDbname>',
  tablename='<yourTablename>',
  username='<yourUsername>',
  password='<yourPassword>',
  endpoint='<yourEndpoint>',
  field_delimiter='|' -- This parameter is optional.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to hologres.

dbname	The name of the database. **Note** If the public schema is not used, you must set tableName to **schema.tableName**.	Yes	N/A.
tablename	The name of the table.	Yes	N/A.
username	The username that is used to access the database.	Yes	N/A.
password	The password that is used to access the database.	Yes	N/A.
endpoint	The virtual private cloud (VPC) endpoint of Hologres.	Yes	For more information, see Endpoints for connecting to Hologres.
field_delimiter	The delimiter used between rows when data is being exported. **Important** Do not insert delimiters in data. This parameter takes effect only when the **bulkload** semantics is used.	No	Default value: "\u0002".
mutateType	The streaming write semantics. For more information, see Streaming data semantics.	No	Default value: insertorignore.

partitionrouter	Specifies whether to write data to a partitioned table.	No	Default value: false. **Note** For Blink 3.6.X, the Hologres streaming sink cannot automatically create partitioned tables. Before Realtime Compute for Apache Flink writes data to a partitioned table, you must manually create a sub-table in Hologres.
ignoredelete	Specifies whether to ignore retraction messages.	No	Default value: false. **Note** This parameter takes effect only after the streaming data semantics is used.

createPartTable	Specifies whether to automatically create a non-existent partitioned table to which data is written based on partition values. **Note** If the partition values contain hyphens (-), partitioned tables cannot be automatically created.	No	• false: Partitioned tables cannot be automatically created. This is the default value. • true: Partitioned tables can be automatically created. **Note** Only Blink versions later than 3.7 support this parameter.

Streaming data semantics

Stream processing, also known as streaming data or event processing, refers to the continuous processing of a series of unbounded data or events. In most cases, the system that processes streaming data or events allows you to specify a reliability pattern or processing semantics to ensure data accuracy. Network or device failures may cause a data loss.

Semantics can be classified into the following types based on the configurations of the Hologres streaming sink that you use and the attributes of the Hologres table:

- **Exactly-once**: The system processes data or events only once even multiple faults occur.
- **At-least-once**: If streaming data or events to be processed are lost, the system transfers the data again from the transmission source. Therefore, the system may process streaming data or events for multiple times. If the first retry succeeds, no further retries are required.

When you use streaming semantics in a Hologres result table, take note of the following points:

- If no primary keys have been configured in the Hologres physical table, the Hologres streaming sink uses the **at-least-once** semantics.
- If primary keys have been configured in the Hologres physical table, the Hologres streaming sink uses the **exactly-once** semantics based on the primary keys. If multiple records with the same primary key are written to the table, you must set the **mutationType** parameter to determine how the result table is updated. This parameter has the following valid values:
  - **insertorignore** (default value): Hologres keeps the first record and discards the subsequent records.
  - **insertorreplace**: Hologres completely replaces the existing record with the one that arrives later.
  - **insertorupdate**: Hologres partially replaces the existing record with the one that arrives later.

> **Note**
>
> - If the **mutationType** parameter is set to **insertorupdate** or **insertorreplace**, the system updates data based on the primary key.
> - The number of columns in the result table defined by Blink can be different from the number of columns in the Hologres physical table. Make sure that the value null can be used to fill the missing columns. Otherwise, an error is returned.

- By default, the Hologres streaming sink can import data to only one non-partitioned table. If the sink imports data into the parent table of a partitioned table, data queries fail even if data import succeeds. To enable data to be automatically written to a partitioned table, you can set the **partitionRouter** parameter to true. Take note of the following points:
  - You must set **tablename** to the name of the parent table.
  - Blink connectors do not automatically create partitioned tables. We recommend that you create a partitioned table before you import data. Otherwise, the data fails to be imported.

Merge data into a wide table

If you need to write data from multiple streaming jobs to one Hologres wide table, you can merge the data into a wide table.

For example, one Flink data stream contains fields A, B, and C, the other contains fields A, D, and E. The Hologres wide table WIDE_TABLE contains fields A, B, C, D, and E, among which field A is the primary key. You can perform the following operations:

1. Execute Flink SQL statements to create two Hologres result tables. One table is used to declare fields A, B, and C, and the other is used to declare fields A, D, and E. Map the two result tables to the Hologres wide table WIDE_TABLE.
2. Parameter settings of the two Hologres result tables:
   - Set **mutatetype** to **insertorupdate**. This indicates that data is updated based on the primary key.
   - Set **ignoredelete** to **true**. This prevents retraction messages from generating Delete requests.
3. Insert data from the two Flink data streams into the two result tables.

> **Note**
>
> Limits:
>
> - The wide table must have a primary key.
> - The data of each stream must contain all the fields in the primary key.
> - If the wide table is a column-oriented table and the requests per second (RPS) value is large, the CPU utilization increases. We recommend that you disable dictionary encoding for the fields in the table.

Data type mapping

Hologres	BLINK
INT	INT

INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

5.6.3.4. Create an AnalyticDB for MySQL V2.0 result table

This topic describes how to create an AnalyticDB for MySQL V2.0 result table. It also describes the mapping between the field data types of AnalyticDB for MySQL and Realtime Compute for Apache Flink.

> **Important**
> - This topic applies only to Blink 1.4.5 and later.
> - You are allowed to define an auto-increment primary key for an AnalyticDB for MySQL V2.0 database. If you want to use the auto-increment primary key, do not declare the auto-increment field in a data definition language (DDL) statement. For example, if you use ID as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written to the ApsaraDB RDS for MySQL database, the value for the auto-increment field is automatically filled.

Introduction to AnalyticDB for MySQL

AnalyticDB for MySQL is a real-time online analytical processing (OLAP) service that is developed by Alibaba Cloud. It is a high concurrency service that has excellent performance in processing large amounts of data. AnalyticDB for MySQL allows you to query and analyze hundreds of billions of data records within milliseconds.

DDL syntax

> **Note** For more information about how to create an AnalyticDB for MySQL V3.0 result table, see Create an AnalyticDB for MySQL V3.0 result table.

In Realtime Compute for Apache Flink, you can use AnalyticDB for MySQL V2.0 to store output data. The following code shows an example:

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='ads',
url='yourDatabaseURL',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>',
batchSize='20'
);
```

> **Note**
> - We recommend that you use the storage registration feature. For more information, see Register an AnalyticDB for MySQL instance.
> - The primary key that is declared in an AnalyticDB for MySQL result table must be consistent with that in an AnalyticDB for MySQL database. The primary key is case-sensitive. Inconsistency may cause the "array index out of bounds" error.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to ads.

url	The Java Database Connectivity (JDBC) URL of a database.	The URL of the AnalyticDB for MySQL database, for example, `url ='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'`. The following descriptions about the parameters in the URL are provided: • Perform the following steps to query the URI: i. Log on to the AnalyticDB for MySQL console. ii. Click the name of the cluster that you want to access to go to the **Basic Information** page. iii. In the **Connection Information** section, view the URL. • The databaseName parameter specifies the name of the AnalyticDB for MySQL database or the name of the AnalyticDB for MySQL instance.
tableName	The name of the table in the database	None.
username	The username that is used to log on to the database.	None.
password	The password that is used to log on to the database	None.
maxRetryTimes	The maximum number of retries for writing data to the table.	Optional. Default value: 10.
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	Optional. Default value: 5000. This value indicates that data deduplication is triggered if the number of input data records in the buffer reaches 5,000.
batchSize	The number of data records that are written at a time.	Optional. Default value: 1000. **Note** If error code 20015 is returned, the batchSize parameter is set to a large value. For AnalyticDB for MySQL databases, the size of data records that are written at a time cannot exceed 1 MB. If the batchSize parameter is set to `1000`, the average size of each data record cannot exceed 1 KB.

batchWriteTimeoutMs	The timeout period for writing data.	Optional. Default value: 5000. Unit: milliseconds. This value indicates that if the number of input data records does not reach the value specified by the batchSize parameter within 5,000 milliseconds, all cached data is written to the result table.
connectionMaxActive	The maximum number of connections in a connection pool.	Optional. Default value: 30.
ignoreDelete	Specifies whether to skip the delete operation.	Default value: false.

Field type mapping

We recommend that you declare the mapping between the field data types of AnalyticDB for MySQL and Realtime Compute for Apache Flink in DDL statements.

Data type of AnalyticDB for MySQL	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
TINYINT	INT
SAMLLINT	
INT	
BIGINT	BIGINT
DOUBEL	DOUBLE
VARCHAR	VARCHAR
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

5.6.3.5. Create a Log Service result table

This topic describes how to create a Log Service result table in Realtime Compute for Apache Flink.

> **Important**
> - This topic applies only to Blink 1.4.5 and later.
> - Log Service result tables support only fields of the VARCHAR type.

What is Log Service?

Log Service is an end-to-end data logging service that is developed by Alibaba Cloud. Log Service allows you to collect, consume, ship, query, and analyze log data in a quick manner. It improves the operations and maintenance (O&M) efficiency and provides the capability to process large amounts of log data. Log Service stores streaming data. Therefore, Realtime Compute for Apache Flink can use Log Service tables as result tables for the processing of streaming data.

DDL syntax

In Realtime Compute for Apache Flink, you can use Log Service to store output data. The following code shows an example.

```
create table sls_stream(
 `name` VARCHAR,
 age BIGINT,
 birthday BIGINT
)with(
 type='sls',
 endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 project='<yourProjectName>',
 logstore='<yourLogstoreName>'
);
```

> **Note**
>
> We recommend that you use the storage registration feature of Log Service. For more information, see Register a Log Service project.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
endPoint	The endpoint of Log Service.	Yes	Endpoints
project	The name of a project.	Yes	N/A.
logstore	The name of the table in the database.	Yes	N/A.
accessId	The AccessKey ID that is used to access Log Service.	Yes	N/A.

accessKey	The AccessKey Secret	Yes	N/A.
topic	An attribute field.	No	This parameter is empty by default. You can use the selected field as the `topic` attribute field.
timestampColumn	An attribute field.	No	This parameter is empty by default. You can use the selected field as the `timestamp` attribute field. The data type of this parameter must be INT. If no field is selected, the current time is used as the attribute field.
source	An attribute field. The source of a log entry. For example, the value of this field can be the IP address of the server where the log entry is generated.	No	This parameter is empty by default. You can use the selected field as the `source` attribute field.
partitionColumn	The partition key column.	No	This parameter is required if the `mode` parameter is set to `partition`.
flushIntervalMs	The interval at which data writing is triggered.	No	Default value: 2000. Unit: milliseconds.
reserveMilliSecond	Specifies whether to reserve the millisecond component in a value of the TIMESTAMP data type.	No	Default value: false. This value indicates that the millisecond component is not reserved. **Note** This parameter is available in Realtime Compute for Apache Flink V2.2.6 and later.

Data type mapping

The following table describes the mapping between the data types of Log Service and Realtime Compute for Apache Flink. We recommend that you declare the mappings in a DDL statement.

Data type of Log Service	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

Sample code

The following sample code demonstrates how to create a Log Service result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE random_input (
  a VARCHAR,
  b VARCHAR) with (
    type = 'random'
);

create table sls_output(
 a varchar,
 b varchar
)with(
 type='sls',
 endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 project='ali-cloud-streamtest',
 logStore='stream-test2'
);

INSERT INTO sls_output
SELECT a, b
FROM random_input;
```

FAQ

Q: How do I specify the **topic** field in a Log Service result table?

A: You can specify the **topic** field as a field in the result table. For example, specify `topic='age'` in the sample code. After the configuration is complete, the value of the `age` field is written into Log Service but Log Service does not write the `age` field into the downstream storage systems.

References

- For more information about Log Service, see What is Simple Log Service?.
- For more information about how to consume Log Service data in Realtime Compute for Apache Flink, see Use Realtime Compute to consume log data.

5.6.3.6. Create a result table

This topic describes how to create a ApsaraMQ for RocketMQ result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause used when you create a ApsaraMQ for RocketMQ result table.

> **Important**
>
> - This topic applies only to Blink 1.4.5 and later.
> - If you need to use ApsaraMQ for RocketMQ that has separate namespaces, use Blink 3.X.

Introduction to ApsaraMQ for RocketMQ

ApsaraMQ for RocketMQ is a professional message middleware that is developed by Alibaba Cloud for commercial use. It is a core product for the enterprise-level Internet architecture. Based on the high-availability distributed cluster technology, ApsaraMQ for RocketMQ provides comprehensive cloud messaging services, including message publishing and subscription, message tracing, resource statistics, message scheduling or delaying, monitoring, and alerts.

CSV format

You can specify ApsaraMQ for RocketMQ tables as result tables for Realtime Compute for Apache Flink to process streaming data. In the following sample code, the DDL statement creates a ApsaraMQ for RocketMQ result table to store streaming data in the CSV format:

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR
) WITH (
type='mq',
endpoint='<yourEndpoint>',
accessID='<yourAccessId>',
accessKey='<yourAccessSecret>',
topic='<yourTopicName>',
producerGroup='<yourGroupName>',
tag='<yourTagName>',
encoding='utf-8',
fieldDelimiter=',',
retryTimes='5',
sleepTimeMs='500'
);
```

Binary format

You can specify ApsaraMQ for RocketMQ tables as result tables for Realtime Compute for Apache Flink to process streaming data. In the following sample code, the DDL statement creates a ApsaraMQ for RocketMQ result table to store streaming data in the binary format:

```
CREATE TABLE source_table (
  commodity VARCHAR
)WITH(
  type='random'
);

CREATE TABLE result_table (
  mess VARBINARY
) WITH (
  type = 'mq',
  endpoint='<yourEndpoint>',
  accessID='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic='<yourTopicName>',
  producerGroup='<yourGroupName>'
);

INSERT INTO result_table
SELECT
CAST(SUBSTRING(commodity,0,5) AS VARBINARY) AS mess
FROM source_table;
```

> **Note**
>
> The `cast(varchar as varbinary)` method is supported only in Blink 2.0 or later. If the Blink version is earlier than 2.0, update the Blink version first. For more information, see Manage Blink versions of a Realtime Compute for Apache Flink cluster deployed in exclusive mode.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to mq.
topic	The name of the ApsaraMQ for RocketMQ topic to which data is written.	N/A.
		Two types of ApsaraMQ for RocketMQ services are provided: internal ApsaraMQ for RocketMQ and public ApsaraMQ for RocketMQ. Select an endpoint based on the type of ApsaraMQ for RocketMQ that you purchase. • For jobs that run on Blink 3.7.10 or later, use Transmission Control Protocol (TCP) endpoints. For more information, see Announcement on the settings of internal TCP endpoints. You can use one of the following methods to obtain the endpoints:

endpoint	The endpoint of ApsaraMQ for RocketMQ	◦ Internal endpoints of ApsaraMQ for RocketMQ that resides in the classic network or a virtual private cloud (VPC) of Alibaba Cloud: Log on to the ApsaraMQ for RocketMQ console. In the left-side navigation pane, click Instances. On the page that appears, find the required instance, and click Details in the Actions column. On the Instance Details page, click the **Endpoints** tab. In the **TCP Endpoint** section, you can view the endpoint that corresponds to **Internal Access**. ◦ Public endpoint of ApsaraMQ for RocketMQ: Log on to the ApsaraMQ for RocketMQ console. In the left-side navigation pane, click Instances. On the page that appears, find the required instance, and click Details in the Actions column. On the Instance Details page, click the **Endpoints** tab. In the **TCP Endpoint** section, you can view the endpoint that corresponds to **Internal Access**. • For jobs that run on Blink of a version earlier than 3.7.10, use the following endpoints: ◦ Internal endpoints of ApsaraMQ for RocketMQ that resides in the classic network or a VPC of Alibaba Cloud: ▪ China (Hangzhou), China (Shanghai), China (Qingdao), China (Beijing), China (Shenzhen), and China (Hong Kong): `onsaddr-internal.aliyun.com:8080` . ▪ Singapore (Singapore): `ap-southeastaddr-internal.aliyun.com:8080` . ▪ UAE (Dubai): `ons-me-east-1-internal.aliyuncs.com:8080` . ▪ India (Mumbai): `ons-ap-south-1-internal.aliyuncs.com:8080` . ▪ Malaysia (Kuala Lumpur): `ons-ap-southeast-3-internal.aliyun.com:8080` . ◦ Public endpoint of ApsaraMQ for RocketMQ: `onsaddr-internet.aliyun.com:80` .

		Important • If you use the ApsaraMQ for RocketMQ connector in Blink of a version earlier than 3.7.10, you must update your Realtime Compute for Apache Flink job to Blink 3.7.10 or later and change the value of the endpoint parameter to the new endpoint of ApsaraMQ for RocketMQ. This reduces the risks of instability or unavailability that are caused by the old endpoint of ApsaraMQ for RocketMQ. For more information, see The endpoints of ApsaraMQ for RocketMQ are unavailable and Realtime Compute for Apache Flink deployments need to be updated to adapt to the change. • **Internal ApsaraMQ for RocketMQ does not support cross-region access**. For example, if your Realtime Compute for Apache Flink service resides in the China (Hangzhou) region but your ApsaraMQ for RocketMQ service resides in the China (Shanghai) region, Realtime Compute for Apache Flink cannot access this ApsaraMQ for RocketMQ service. • By default, Realtime Compute for Apache Flink clusters in exclusive mode cannot access the Internet. If you want to access the Internet, configure a NAT gateway. • The network security policies of Alibaba Cloud dynamically change. As a result, network connection issues may occur when Realtime Compute for Apache Flink connects to the public ApsaraMQ for RocketMQ. We recommend that you use the internal ApsaraMQ for RocketMQ service.
accessID	AccessKey ID	N/A.
accessKey	AccessKey Secret	N/A.
producerGroup	Specifies the name of the producer group to which messages are written.	N/A.
tag	The message tag.	Optional. This parameter is empty by default.
fieldDelimiter	The field delimiter.	Optional. Default value: `\u0001`. The delimiter varies based on the following modes: • In read-only mode, the `\u0001` delimiter is used. `\u0001` is invisible in read-only mode. • In edit mode, the `^A` delimiter is used.
encoding	The encoding format.	Optional. Default value: `utf-8`.
retryTimes	The number of retries for writing data to the table.	Optional. Default value: 10.
sleepTimeMs	The retry interval.	Optional. Default value: 1000. Unit: milliseconds.

instanceID	The ID of a ApsaraMQ for RocketMQ instance.	• If the ApsaraMQ for RocketMQ instance does not have a separate namespace, the **instanceID** parameter cannot be used. • If the ApsaraMQ for RocketMQ instance has a separate namespace, the **instanceID** parameter is required.

5.6.3.7. Create a Tablestore result table

This topic describes how to create a Tablestore result table in Realtime Compute for Apache Flink. It also describes the mappings between the field data types of Tablestore and Realtime Compute for Apache Flink.

> **Important**
>
> This topic applies only to Blink 1.4.5 and later.

Introduction to Tablestore

Tablestore is a distributed NoSQL database service built on the Apsara distributed operating system of Alibaba Cloud. Tablestore adopts data sharding and load balancing technologies to scale out and handle concurrent transactions. You can use Tablestore to store and query a large amount of structured data in real time.

DDL syntax

In Realtime Compute for Apache Flink, you can use Tablestore to store output data. The following code shows an example:

```
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**
>
> - We recommend that you use the storage registration feature. For more information, see Register a Tablestore instance.
> - The value of the valueColumns parameter cannot be a declared primary key.
> - The declared Tablestore result table must contain at least one attribute column and the primary key column.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to ots.
instanceName	The name of a Tablestore instance.	None.
tableName	The name of the table in the database	None.
endPoint	The endpoint of the instance.	For more information, see Endpoints.
accessId	AccessKey ID	None.
accessKey	AccessKey Secret	None.
valueColumns	The name of a column to be inserted.	Separate multiple column names with commas (,), for example, `'ID,NAME'`.
bufferSize	The maximum number of data records that can be stored in the buffer before deduplication is triggered.	Optional. Default value: 5000. This value indicates that deduplication is triggered if the number of input data records in the buffer reaches 5,000. **Note** Realtime Compute for Apache Flink removes data record duplicates based on the primary key of the Tablestore result table. You can set bufferSize to the number of data record duplicates to be removed. Then, Realtime Compute for Apache Flink writes the data records after duplicates are removed. You can set batchSize to the number of data records to be written at a time.
batchWriteTimeoutMs	The write timeout period.	Optional. Default value: 5000. Unit: milliseconds. This value indicates that if the number of input data records does not reach the value specified by the batchSize parameter within 5,000 milliseconds, all cached data is written into the result table.
batchSize	The number of data records that can be written at a time.	Optional. Default value: 100.

retryIntervalMs	The retry interval.	Optional. Default value: 1000. Unit: milliseconds.
maxRetryTimes	The maximum number of retries for writing data to a table.	Optional. Default value: 100.
ignoreDelete	Specifies whether to ignore DELETE operations.	Default value: false.

Field type mapping

Data type of Tablestore	Data type of Realtime Compute for Apache Flink
INTEGER	BIGINT
STRING	VARCHAR
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

> **Note**
>
> You must define a `primary key` in a Tablestore result table. Output data is appended to the Tablestore result table to update the result. For more information about update methods, see Update type.

5.6.3.8. Create an ApsaraDB RDS result table

This topic describes how to create an ApsaraDB RDS result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause and data type mapping used when you create an ApsaraDB RDS result table.

> **Important**
>
> Realtime Compute for Apache Flink does not allow you to use ApsaraDB RDS for MySQL V8.0 by using the storage registration method. To use ApsaraDB RDS for MySQL V8.0, we recommend that you configure a plaintext AccessKey pair. For more information about the storage registration method, see Overview.

Introduction to ApsaraDB RDS

ApsaraDB RDS is a stable, reliable, and scalable online database service. ApsaraDB RDS supports a wide range of database engines, such as MySQL, SQL Server, PostgreSQL, and Postgres Plus Advanced Server (PPAS), based on Apsara Distributed File System and high-performance storage services. ApsaraDB RDS provides comprehensive solutions for database operations and maintenance (O&M), such as disaster recovery, data backup, data recovery and restoration, monitoring, and data migration.

> **Note**
>
> ApsaraDB RDS, Distributed Relational Database Service (DRDS), and PolarDB use the same parameters in the WITH clause. If you want to use an ApsaraDB RDS, DRDS, or PolarDB table as a result table, make sure that a real table exists.

DDL syntax

The following sample code shows how to create an ApsaraDB RDS or DRDS result table. Only ApsaraDB RDS for MySQL is supported.

```
CREATE TABLE rds_output(
   id INT,
   len INT,
   content VARCHAR,
   PRIMARY KEY (id,len)
) WITH (
   type='rds',
   url='<yourDatabaseURL>',
   tableName='<yourDatabaseTable>',
   userName='<yourDatabaseUserName>',
   password='<yourDatabasePassword>'
);
```

> **Note**
>
> - In Realtime Compute for Apache Flink, each row of output data is converted to a line of SQL statement and then written and executed in the destination database. If you want to write multiple rows of output data at a time, you must add `?rewriteBatchedStatements=true` to the end of the URL. This improves the system performance.
> - You can define an auto-increment primary key for the ApsaraDB RDS for SQL database that stores the result table. If you want to use the auto-increment primary key, do not declare the auto-increment field in the DDL statement. For example, if you use ID as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written to the ApsaraDB RDS for MySQL database, the value for the auto-increment field is automatically filled.
> - We recommend that you use the storage registration method to connect to the database. For more information, see Register an ApsaraDB for RDS instance.
> - You must declare at least one non-primary key in the DDL statement. Otherwise, an error is returned.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to `rds`.
url	The Java Database Connectivity (JDBC) URL of the database.	Yes	Set the value in the `jdbc:mysql://<Internal endpoint>/<databaseName>` format. Replace **databaseName** with the name of your database. For more information about the internal endpoint, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for MySQL instance.
tableName	The name of the table.	Yes	N/A.
userName	The username that is used to access the database.	Yes	N/A.
password	The password that is used to access the database.	Yes	N/A.
maxRetryTimes	The maximum number of retries that are allowed to write data to the table.	No	Default value: 10.
batchSize	The number of data records that is written at a time.	No	Default value: 4096.
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	No	Default value: 10000.
flushIntervalMs	The interval at which the cache is cleared.	No	Default value: 2000. Unit: milliseconds. This value indicates that if the number of input data records does not reach the value specified by the bufferSize parameter within 2,000 milliseconds, all cached data is written to the result table.

excludeUpdateColumns	Specifies whether to ignore the update of the specified field.	No	This parameter is empty by default. If it is empty, the primary key column is not updated. When data with the same primary key is updated, the specified columns are not updated.
ignoreDelete	Specifies whether to skip the delete operation.	No	Default value: false. This value indicates that the delete operation is supported.
partitionBy	The columns based on which Realtime Compute for Apache Flink performs hash partitioning.	No	This parameter is empty by default. Before Realtime Compute for Apache Flink writes data to the sink node, Realtime Compute for Apache Flink performs hash partitioning based on the value of this parameter. The data then flows to the relevant hash node.

Data type mapping

Data type of ApsaraDB RDS	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

VARCHAR	VARCHAR
VARBINARY	VARBINARY

JDBC 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, you must set this parameter to true.	false	1.1g
characterEncoding	The character encoding format, such as GB2312 or GBK. If useUnicode is set to true, you must specify a character encoding format.	false	1.1g
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 the database is read-only after it is automatically reconnected.	true	3.0.12
maxReconnects	Specifies the maximum number of reconnection attempts allowed if the autoReconnect parameter is set to true.	3	1.1
initialTimeout	Specifies the interval between two reconnection attempts if the autoReconnect parameter is set to true. Unit: seconds.	2	1.1
connectTimeout	Specifies the timeout period when you use a socket connection to access the database server. Unit: milliseconds. Default value: 0. This value indicates that the connection never times out. This parameter applies to JDK V1.4 and later.	0	3.0.1

socketTimeout	The timeout period for a socket-based read or write operation. Unit: milliseconds. Default value: 0. This value indicates that the read or write operation never times out.	0	3.0.1

Sample code

The following sample code shows how to create an ApsaraDB RDS result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'random'
);

CREATE TABLE rds_output(
   id INT,
   len INT,
   content VARCHAR,
   PRIMARY KEY (id,len)
) WITH (
   type='rds',
   url='<yourDatabaseURL>',
   tableName='<yourDatabaseTable>',
   userName='<yourDatabaseUserName>',
   password='<yourDatabasePassword>'
);

INSERT INTO rds_output
SELECT id, len, content FROM source;
```

FAQ

- Q: When output data is written to an ApsaraDB RDS result table, is a new data record inserted into the table or is the result table updated based on the primary key value?

  A: If a primary key is defined in the DDL statement, the following statement is executed to write output data: `INSERT INTO tablename(field1,field2, field3, ...) VALUES(value1, value2, value3, ...) ON DUPLICATE KEY UPDATE field1=value1,field2=value2, field3=value3, ...;`. If the primary key value in the output data exists in the table, the matching record is updated. Otherwise, the output data is inserted as a new record. If no primary key is defined in the DDL statement, the `INSERT INTO` statement is executed to insert the output data.

- Q: How do I perform GROUP BY operations by using the unique index of an ApsaraDB RDS result table?

  A: Use the following method to resolve the issue:

  - You must declare the unique index in the GROUP BY clause in your Realtime Compute for Apache Flink job.
  - An ApsaraDB RDS table has only one auto-increment primary key, which cannot be declared as a primary key in a Realtime Compute for Apache Flink job.

5.6.3.9. Create a MaxCompute result table

This topic describes how to create a MaxCompute result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, data type mappings, and FAQ involved when you create a MaxCompute result table.

> **Important**
>
> - Only Blink 1.5.1 and later support MaxCompute result tables.
> - A clustered table of MaxCompute cannot be used as a result table.

Principles

The MaxCompute sink works in two phases:

1. Writes data. The MaxCompute sink calls an interface in the MaxCompute SDK to write data to the buffer. Then, the sink uploads data to the temporary files of MaxCompute at the specified interval or when the data size in the buffer exceeds 64 MB.
2. Commits sessions. When a task creates checkpoints, the MaxCompute sink calls the Tunnel commit method to commit sessions and moves temporary files to the data directory of the MaxCompute table. Then, the MaxCompute sink modifies the metadata.

> **Note**
>
> The commit method does not provide atomicity. Therefore, the MaxCompute sink supports at-least-once delivery instead of exactly-once delivery.

DDL syntax

In Realtime Compute for Apache Flink, you can use MaxCompute to store output data. The following code shows an example:

> **Note**
>
> The names, sequence, and types of the fields that are defined in the data definition language (DDL) statement must be the same as those in the MaxCompute physical table. Otherwise, the queried data in the MaxCompute physical table may be `/n` .

```
create table odps_output(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  type = 'odps',
  endPoint = '<YourEndPoint>',
  project = '<YourProjectName>',
  tableName = '<YourtableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to `odps`.
endPoint	The endpoint of MaxCompute.	Yes	For more information, see Endpoints.
tunnelEndpoint	The endpoint of MaxCompute Tunnel.	Yes	For more information, see Endpoints. **Note** This parameter is required if MaxCompute is deployed in a virtual private cloud (VPC).
project	The name of the MaxCompute project.	Yes	N/A.
tableName	The name of the table.	Yes	N/A.
accessId	The AccessKey ID that is used to access MaxCompute.	Yes	N/A.
accessKey	The AccessKey secret that is used to access MaxCompute.	Yes	N/A.

partition	The name of a partition.	No	This parameter is required if a partitioned table exists. • Static partitions For example, `` `partition`='ds=20180905' `` indicates that data is written to the `ds=20180905` partition. • Dynamic partition (available in Blink 3.2.1 and later) If the partition values are not displayed in plaintext mode, data is written to different partitions based on the values of the partition key columns specified in the data. For example, `` `partition`='ds' `` indicates that data is written to partitions based on the value of the `ds` field. If you want to create multi-level dynamic partitions, make sure that the sequence of the partition fields in the WITH clause and DDL statement of the MaxCompute result table is consistent with the field sequence of the MaxCompute physical table. Multiple partition fields are separated by commas (,). **Note** ◦ In the CREATE TABLE statement, you must explicitly specify the dynamic partition key column that you use to create dynamic partitions. ◦ If the partition field for dynamic partitions is left empty and the value of the ds field is `null` or `''`, the output varies based on the Blink version: ▪ For Blink 3.2.1 and earlier, a NullPointerException (NPE) error is returned. ▪ For Blink 3.2.2 and later, a partition with **ds=null** is created.

flushIntervalMs	The flush interval for the buffer of a writer in MaxCompute Tunnel. Unit: milliseconds. The MaxCompute sink inserts data into its buffer and then writes the data from the buffer into the destination MaxCompute table at an interval specified by the flushIntervalMs parameter or when the buffer overflows.	No	Default value: 30000. Unit: milliseconds. **Note** This parameter is available in Blink 3.6.0 and later.
partitionBy	The columns based on which hash shuffling is implemented. Before data is written to a sink node, the system implements hash shuffling based on the value of this parameter. This way, data flows to the related sink nodes.	No	The system performs hash shuffling based on multiple columns. The column names are separated by commas (,). By default, this parameter is empty. **Note** This parameter is available in Blink 3.6.0 and later.
isOverwrite	Specifies whether to clear the result table before data is written to a sink node.	No	• For versions earlier than Blink 3.2, the default value is **true**. • For Blink 3.2 and later, the default value is **false**. When you declare a MaxCompute result table for a streaming job, you must set the **isOverwrite** parameter to **false**. Otherwise, an error is returned during compilation. **Note** For Blink 3.2 and later, you can change the value of the **isOverwrite** parameter to **true**. For Blink versions earlier than 3.2, if you want to change the value of the **isOverwrite** parameter, you must upgrade the Blink version first.

dynamicPartitionLimit	The maximum number of partitions.	No	The default value is 100. A map in the memory maintains the mappings between the existing partitions to which data is written and the Tunnel service or the writer. If the map size exceeds the value of the **dynamicPartitionLimit** parameter, the system reports the following error: `Too many dynamic partitions: 100, which exceeds the size limit: 100` .

Data type mappings

Data type of MaxCompute	Data type of Realtime Compute for Apache Flink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
STRING	VARCHAR
DECIMAL	DECIMAL

Sample code

The following sample code shows how to create a MaxCompute result table in a Realtime Compute for Apache Flink job.

- Write data to a static partition

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'random'
);

create table odps_sink (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'odps',
   endPoint = '<yourEndpoint>',
   project = '<yourProjectName>',
   tableName = '<yourTableName>',
   accessId = '<yourAccessId>',
   accessKey = '<yourAccessPassword>',
   `partition` = 'ds=20180418'
);

INSERT INTO odps_sink
SELECT
   id, len, content
FROM source;
```

- Write data to a dynamic partition

```
CREATE TABLE source (
  id INT,
  len INT,
  content VARCHAR,
  c TIMESTAMP
) with (
  type = 'random'
);

create table odps_sink (
  id INT,
  len INT,
  content VARCHAR,
  ds VARCHAR --The partition key column that you use to create dynamic partitions mus
t be explicitly specified in the CREATE TABLE statement.
) with (
  type = 'odps',
  endPoint = '<yourEndpoint>',
  project = '<yourProjectName>',
  tableName = '<yourTableName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessPassword>',
  `partition`='ds' --The partition value is not provided. This means that data is wri
tten to different partitions based on the value of the ds field.
);

INSERT INTO odps_sink
SELECT
   id,
   len,
   content,
   DATE_FORMAT(c, 'yyMMdd') as ds
FROM source;
```

FAQ

- Q: What do the endPoint and tunnelEndpoint parameters mean in the Alibaba Cloud public cloud? What happens if the two parameters are incorrectly configured?

  A: For more information about the endPoint and tunnelEndpoint parameters, see Endpoints. If the configuration of these two parameters is incorrect in a VPC, one of the following task exceptions may occur.

  - If the endPoint parameter is incorrectly configured, the task stops at a progress of 91%.
  - If the tunnelEndpoint parameter is incorrectly configured, the task fails.

- Q: Does Realtime Compute for Apache Flink clear a MaxCompute result table before it writes data to the result table in stream mode when **isOverwrite** is set to **true**?

  A: Blink versions earlier than 3.2 support this feature. Blink 3.2 and later do not support this feature.

  If the **isOverwrite** parameter is set to **true**, Realtime Compute for Apache Flink clears a MaxCompute result table before it writes data to the result table. Realtime Compute for Apache Flink clears data from the existing result table or the result partition each time a job is started or resumed, or before Realtime Compute for Apache Flink writes data to the result table.

- For Blink versions earlier than 3.2, the default value of **isOverwrite** is **true** and cannot be changed. Data may be lost after a streaming job is suspended or resumed.
- For Blink 3.2 and later, the default value of **isOverwrite** is **false**. If you declare a MaxCompute result table for a streaming job, you must set the **isOverwrite** parameter to **false**. Otherwise, an error is returned during compilation. MaxCompute result tables in stream mode support the at-least-once data security mechanism. If a job fails, duplicate data may be generated.

- Q: What do I do if the error message "ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'" appears when a job is running?

  A: This error occurs because the user identity information specified in the MaxCompute DDL statements cannot be used to access MaxCompute. Therefore, you must use an Alibaba Cloud account, a RAM user, or a RAM role to authenticate the user identity. For more information, see User authentication.

5.6.3.10. Create an ApsaraDB for HBase result table

This topic describes how to create an ApsaraDB for HBase result table in Realtime Compute for Apache Flink.

> **Important**
> - This topic applies only to Realtime Compute for Apache Flink in exclusive mode.
> - Blink versions earlier than 3.3.0 support only HBase Standard Edition.
> - Blink 3.3.0 and later versions support both HBase Standard Edition and HBase Enhanced Edition.
> - Blink 3.5.0 and later versions support switchover between primary and secondary ApsaraDB for HBase databases for data writing.
> - ApsaraDB for HBase result tables in Realtime Compute for Apache Flink do not support self-managed open source HBase.

DDL syntax

In Realtime Compute for Apache Flink, you can use ApsaraDB for HBase to store output data.

- The following sample code demonstrates how to create an ApsaraDB for HBase result table of HBase Standard Edition:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '2',
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- The following sample code demonstrates how to create an ApsaraDB for HBase result table

of HBase Enhanced Edition:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    endPoint = '<host:port>', ----The Java API URL that is used to access the Enhanced Edition of an ApsaraDB for HBase database.
    userName  = 'root', -- The username that is used to access the ApsaraDB for HBase database.
    password = 'root', --The password that is used to access the ApsaraDB for HBase database.
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- The following sample code demonstrates how to create an ApsaraDB for HBase dimension table of HBase Enhanced Edition in Blink 3.5.0 or later:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '<host:port>', ----The Java API URL that is used to access the Enhanced Edition of an ApsaraDB for HBase database.
    userName  = 'root', --The username that is used to access the ApsaraDB for HBase database.
    password = 'root', --The password that is used to access the ApsaraDB for HBase database.
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- The following sample code demonstrates how to create an ApsaraDB for HBase dimension table in Blink 3.5.0 or later that supports switchover between primary and secondary ApsaraDB for HBase databases for data writing:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '<host:port>', ---- The URL that is used to access ApsaraDB for HBase databases in high availability (HA) mode.
   haClusterID = 'ha-xxx', ---- The instance ID of ApsaraDB for HBase databases in HA mode.
    userName  = 'root', --The username that is used to access the ApsaraDB for HBase database.
    password = 'root', --The password that is used to access the ApsaraDB for HBase database.
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

Note

- You can define multiple fields for the primary key. Multiple fields are separated with the value of `rowkeyDelimiter`. The default value is a colon (`:`).
- When you undo the deletion operation in ApsaraDB for HBase, if a column stores multiple versions of a value, all the versions of the value are deleted.
- The connection parameters are different between ApsaraDB for HBase Standard Edition and ApsaraDB for HBase Enhanced Edition.
  - In ApsaraDB for HBase Standard Edition, the connection parameter is `zkQuorum`.
  - In ApsaraDB for HBase Enhanced Edition, the connection parameter is `endPoint`.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to cloudhbase.
zkQuorum	The ZooKeeper address that is configured for the ApsaraDB for HBase cluster.	Yes	You can view the configuration related to hbase.zookeeper.quorum in the hbase-site.xml file. **Note** This parameter takes effect in only ApsaraDB for HBase Standard Edition.

zkNodeParent	The path of the cluster configured on the ZooKeeper servers.	No	You can view the relevant configurations of hbase.zookeeper.quorum in the hbase-site.xml file. **Note** This parameter takes effect only in ApsaraDB for HBase Standard Edition.
endPoint	The name of the region where your ApsaraDB for HBase instance is deployed.	Yes	You can obtain the value of this parameter from the console of your ApsaraDB for HBase instance. **Note** This parameter takes effect only in ApsaraDB for HBase Enhanced Edition.
userName	The username that is used to access an ApsaraDB for HBase database.	No	This parameter takes effect only in ApsaraDB for HBase Enhanced Edition.
password	The password that is used to access an ApsaraDB for HBase database.	No	This parameter takes effect only in ApsaraDB for HBase Enhanced Edition.
tableName	The name of the ApsaraDB for HBase table.	Yes	None.
columnFamily	The name of the column family.	Yes	Only the same column family can be inserted.
maxRetryTimes	The maximum number of retries.	No	Default value: 10. **Note** This parameter is available only in Realtime Compute for Apache Flink V3.2.3 and later.
bufferSize	The maximum number of data records that can be stored in the buffer before deduplication is triggered.	No	Default value: 5000.
batchSize	The number of data records that can be written at a time.	No	Default value: 100. **Note** We recommend that you set this parameter to a value that ranges from 200 to 300. A large value of batchSize may cause an out-of-memory (OOM) error for the task.
flushIntervalMs	The interval at which the buffer is cleared to reduce the latency of writing data into ApsaraDB for HBase.	No	Default value: 2000. Unit: milliseconds.

writePkValue	Specifies whether to write the primary key value.	No	Default value: false.
stringWriteMod	Specifies whether to insert data as the STRING type.	No	Default value: false.
rowkeyDelimiter	The delimiter of **rowKey**.	No	The default delimiter is a colon (:).
isDynamicTable	Specifies whether the table is a dynamic table.	No	Default value: false.
haClusterID	The ID of an ApsaraDB for HBase instance in HA mode.	No	This parameter is required only when you access zone-disaster recovery instances.

Dynamic table

Some result data of Realtime Compute for Apache Flink is used as a dynamic column based on the value of a column and written to ApsaraDB for HBase. In the following example, the turnover per hour is used as a dynamic column data in ApsaraDB for HBase.

rowkey	cf:0	cf:1	cf:2
20170707	100	cf:1	300

If **isDynamicTable** is set to true, the table is an ApsaraDB for HBase table that supports dynamic columns.

A dynamic table can contain only three columns, such as ROW_KEY, COLUMN, and VALUE. The second column that is COLUMN in this example is a dynamic column. Other parameters in the dynamic table are the same as those in the WITH clause of ApsaraDB for HBase.

> **Note** When a dynamic table is used, all data types must be converted to the STRING type before data is written into ApsaraDB for HBase.

```
CREATE TABLE stream_test_hotline_agent (
  name varchar,
  age varchar,
  birthday varchar,
  primary key (name)
) WITH (
  type = 'cloudhbase',
  ...
  columnFamily = 'cf',
  isDynamicTable ='true'
);
```

> **Note**
> - In the preceding declaration, `birthday` is inserted into the `cf:age` column with ROW_KEY of `name`. For example, `(wang,18,2016-12-12)` is inserted into the row for which the ROW_KEY value is `wang` and the `cf:18` column.
> - In the DDL statement, you must declare ROW_KEY as the primary key and declare the following fields in the descending order: ROW_KEY, COLUMN, and VALUE. In this example, ROW_KEY is `name`, COLUMN is `age`, and VALUE is `birthday`.

Sample code

The following sample code demonstrates how to create an ApsaraDB for HBase result table in a Realtime Compute for Apache Flink job:

```
create table source (
  id   TINYINT,
  name BIGINT
) with (
  type = 'random'
);

create table sink (
  id    TINYINT,
  name  BIGINT,
  primary key (id)
) with (
  type = 'cloudhbase',
  zkQuorum = '<yourZkQuorum>',
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>'
);

INSERT INTO sink
SELECT id, name FROM source;
```

FAQ

Why is the error `cloudHbase update error, No columns to insert for #10 item` reported when a job for an ApsaraDB for HBase result table is running?

The column data for a single record that is written into the ApsaraDB for HBase result table cannot all be null. The column data excludes ROW_KEY. Before you use Realtime Compute for Apache Flink to write data into the ApsaraDB for HBase result table, filter out all null data.

5.6.3.11. Create an Elasticsearch result table

This topic describes how to create an Elasticsearch result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause used when you create an Elasticsearch result table.

> **Important** This topic applies only to Blink 3.2.2 and later.

DDL syntax

In Realtime Compute for Apache Flink, you can use Elasticsearch to store output data. The following sample code shows how to create an Elasticsearch result table.

```
 CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1)
)WITH(
  type ='elasticsearch',
  endPoint = 'http://es-cn-mp****.public.elasticsearch.aliyuncs.com:****',
  accessId = '<yourUsername>',
  accessKey = '<yourPassword>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

> **Note**
> - Elasticsearch supports data updates based on the PRIMARY KEY field. You can use only one field as the PRIMARY KEY field.
> - If you specify the PRIMARY KEY field, values in the field are used as document IDs.
> - If you do not specify the PRIMARY KEY field, document IDs are randomly generated. For more information, see Index API.
> - In full update mode, later documents overwrite earlier documents.
> - In incremental update mode, the system updates the fields based on the field values you entered.
> - By default, all updates use the UPSERT syntax, which means to insert or update data.

Parameters in the WITH clause (general configurations)

Parameter	Description	Required	Default value
type	The type of the connector.	Yes	elasticsearch
endPoint	The endpoint of an Elasticsearch cluster, such as http://127.0.0.1:9211.	Yes	N/A
accessId	The AccessKey ID that is used to access the Elasticsearch cluster. **Note** If you use the Kibana plug-in to access the Elasticsearch cluster, enter the Kibana logon ID.	Yes	N/A
accessKey	The AccessKey secret that is used to access the Elasticsearch cluster. **Note** If you use the Kibana plug-in to access the Elasticsearch cluster, enter the Kibana logon password.	Yes	N/A

index	The name of the index.	Yes	N/A
typeName	The type of a document.	Yes	`_doc`
bufferSize	The maximum number of data records that can be stored in the buffer before data is deduplicated.	No	1000
maxRetryTimes	The maximum number of retries for writing data to a table.	No	30
timeout	The read timeout period. Unit: milliseconds.	No	600000
discovery	Specifies whether node discovery is enabled. If this feature is enabled, the client refreshes the server list every 5 minutes.	No	false
compression	Specifies whether to compress request bodies in the gzip format.	No	true
multiThread	Specifies whether to enable multithreading for JestClient.	No	true
ignoreWriteError	Specifies whether to ignore write exceptions.	No	false
settings	The settings used to create indexes.	No	N/A
updateMode	The update mode used after the primary key is specified. • full: Full data is overwritten. • inc: Incremental data is added.	No	full

Parameters in the WITH clause (dynamic indexing)

Parameter	Description	Required	Remarks
dynamicIndex	Specifies whether to enable dynamic indexing.	No	Valid values: • true: Dynamic indexing is enabled. • false: Dynamic indexing is disabled. This is the default value.
indexField	The field name of the index.	This parameter is required if the **dynamicIndex** parameter is set to true.	Only the TIMESTAMP (in seconds), DATE, and LONG data types are supported.

indexInterval	The interval at which an index is changed.	This parameter is required if the **dynamicIndex** parameter is set to true.	Valid values: • d: one day. This is the default value. • m: one month. • w: one week.
mapping	The mappings between the types and formats of fields in a document configured when dynamic indexing is enabled. The following example shows how to set the mapping between the type and format of the sendTime field: `{ "properties": { "sendTime": { "type": "date", "format": "yyyy-MM-dd HH:mm:ss" } } }`	No	This parameter is empty by default. **Note** • Blink 3.7.0 and later support this parameter. • Elasticsearch V7.0 does not support this parameter.

Note

- Only Blink 2.2.7 and later support the dynamic indexing feature.
- After dynamic indexing is enabled, the `index` name in the basic configuration is used as the unified alias for indexes created subsequently. An alias can correspond to multiple indexes.
- Actual index names that correspond to different values of `indexInterval` :
  - **d -> Alias "yyyyMMdd"**
  - **m -> Alias "yyyyMM"**
  - **w -> Alias "yyyyMMW"**
- You can use Index API to change a single actual index, but the alias supports only the `GET` method. For more information about how to change the alias, see Index Aliases.

Sample code

The following sample code shows how to create a dynamic index.

```
CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 TIMESTAMP,
  PRIMARY KEY(field1)
)WITH(
  type ='elasticsearch',
  endPoint = 'http://es-cn-mp****.public.elasticsearch.aliyuncs.com:****',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>',
  dynamicIndex = 'true',
  indexField = 'field3',
  indexInterval = 'd'
);
```

5.6.3.12. Create a TSDB result table

This topic describes how to create a Time Series Database (TSDB) result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause that is used when you create a TSDB result table.

> **Important**
> - This topic applies only to Blink 2.0 and later.
> - To reference a TSDB result table in Realtime Compute for Apache Flink, you must configure a whitelist that controls access to storage resources. For more information, see Configure a whitelist for accessing storage resources.

Introduction to TSDB

Alibaba Cloud TSDB is a database service that supports efficient reads and writes, compressed storage, and real-time computing for time series data. TSDB is widely used in Internet of Things (IoT) and Internet fields to implement real-time monitoring, forecasting, and alerting on devices and services.

DDL syntax

In Realtime Compute for Apache Flink, you can use TSDB to store output data. The following code shows an example:

```
CREATE TABLE stream_test_hitsdb (
    metric      VARCHAR,
    `timestamp` INTEGER,
    `value`     DOUBLE,
    tagk1       VARCHAR,
    tagk2       VARCHAR,
    tagk3       VARCHAR
) WITH (
    type='hitsdb',
    host='<yourHostName>',
    virtualDomainSwitch = 'false',
    httpConnectionPool = '20',
    batchPutSize = '1000'
);
```

Default format for tables:

- Column 0: metric (VARCHAR).
- Column 1: timestamp (INTEGER). Unit: seconds.
- Column 2: value (DOUBLE).
- Columns 3 to N: tag keys. The field names in the time series database are used as the tag keys.

> **Note**
> - You can specify multiple **tag** columns.
> - You must declare the following fields: **metric**, **timestamp**, and **value**. The names, sequence, and data types of the fields must be the same as those in TSDB.
> - For more information about parameter settings,see Write data.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to `hitsdb`.
host	The IP address or the domain name that is mapped to the virtual IP address (VIP) of the TSDB instance.	Enter the hostname of the TSDB instance. For more information, see Connect to the instance.
port	The port number that is used to access the TSDB instance.	Default value: 8242.
virtualDomainSwitch	Specifies whether to use VIPServer.	Default value: false. If you need to use VIPServer, set this parameter to true.

httpConnectionPool	The maximum number of connections in an HTTP connection pool.	Default value: 10.
httpCompress	Specifies whether to use GZIP compression.	Default value: false. This value indicates that GZIP compression is not used.
httpConnectTimeout	The timeout period for an HTTP connection.	Default value: 0.
ioThreadCount	The number of I/O threads.	Default value: 1.
batchPutBufferSize	The buffer size.	Default value: 10000.
batchPutRetryCount	The maximum number of retries for writing data to the result table.	Default value: 3.
batchPutSize	The maximum number of data records that can be submitted at a time.	Default value: 500.
batchPutTimeLimit	The maximum duration for which a data record can be stored in the buffer.	Default value: 200. Unit: milliseconds.
batchPutConsumerThreadCount	The number of serialized threads.	Default value: 1.

Models for writing data from Realtime Compute for Apache Flink to TSDB

In Blink 3.2 and later, Realtime Compute for Apache Flink can write data to TSDB by using one of the following models:

- Write single-value data points where no tags are included. To use this model, use the specified schema that consists of three fields. The names of these fields cannot be changed. You must use the following schema format for this model:

```
metric,timestamp,value
```

- Write single-value data points where tags are included. To use this model, use the specified schema. In the schema, the names of the following fields cannot be changed: metric, timestamp, and value. You can specify the tag names based on your business requirements. You must use the following schema format for this model:

```
metric,timestamp,value,tagKey1,....,tagKeyN
```

- Write single-value data points where the number of tags is unknown. To use this model, use the specified schema that consists of four fields. The names of the four fields cannot be changed. You must use the following schema format for this model:

  ```
  metric,timestamp,value,tags
  ```

  In the schema, specify the value of the tags parameter as a JSON string. The JSON string allows you to specify an unknown number of tags. If you do not use the JSON string, you must specify a fixed number of tags for the Blink table schema. In the schema, specify the JSON string in the following format:

  ```
  {"tagKey1":"tagValue1","tagKey2":"tagValue2",……,"tagKeyN":"tagValueN"}
  ```

- Write multi-value data points where tags are not included. You must use the following schema format for this model:

  ```
  metric,timestamp,field_name1,field_name2,……,field_nameN
  ```

  The names of the metric and timestamp fields cannot be changed. For multi-value fields, the field_ prefix is used for each field to distinguish fields from tags and to support single-value data writes. For example, when the `field_name1` field is written to TSDB, the prefix `field_` is automatically removed. In the preceding schema, name1 and name2 are the names of the multi-value fields. For the preceding schema, the following format is used to write data to TSDB:

  ```
  metric,timestamp,name1,name2,……,nameN
  ```

- Write multi-value data points where tags are included. To use this model, use the specified schema. In the schema, the names of the following fields cannot be changed: metric and timestamp. You can specify the tag names based on your business requirements. You must use the following schema format for this model:

  ```
  metric,timestamp,tagKey1,....,tagKeyN,field_name1,field_name2,……,field_nameN
  ```

- Write single-value data points where the number of tags is unknown. You must use the following schema format for this model:

  ```
  metric,timestamp,tags,field_name1,field_name2,……,field_nameN
  ```

  The content of tags is a JSON string shown in the following example. Therefore, the limit that the Blink table schema requires a fixed number of tags does not apply.

  ```
  {"tagKey1":"tagValue1","tagKey2":"tagValue2",……,"tagKeyN":"tagValueN"}
  ```

FAQ

Q: Why does the following error occur during a failover "The values of the LONG data type cannot be converted into the values of the INT data type"?

A: Blink versions earlier than 2.2.5 support only the INT data type. Blink 2.2.5 and later support the BIGINT data type.

5.6.3.13. Create a Message Queue for Apache Kafka result table

This topic describes how to create a Message Queue for Apache Kafka result table in Realtime Compute for Apache Flink. It also describes the mapping between the values of the type parameter and Kafka versions.

> **Important**
> - This topic applies only to Realtime Compute for Apache Flink V2.0 and later.
> - This topic applies only to Realtime Compute for Apache Flink in exclusive mode.
> - Data of a Message Queue for Apache Kafka result table can be written into a self-managed Kafka cluster. Before data is written, you must pay attention to the mapping between the values of the type parameter and Kafka versions, and the network configurations of the self-managed Kafka cluster and the Realtime Compute for Apache Flink cluster.

Introduction to the Message Queue for Apache Kafka result table

Message Queue for Apache Kafka is a distributed, high-throughput, and scalable message queue service provided by Alibaba Cloud. This service is widely used in big data fields, such as log collection, monitoring data aggregation, streaming data processing, and online and offline data analysis. Realtime Compute for Apache Flink allows you to create source tables and result tables of Message Queue for Apache Kafka for the processing of streaming data.

DDL syntax

The following example demonstrates how to create a Message Queue for Apache Kafka result table in a data definition language (DDL) statement.

```
create table sink_kafka (
  messageKey VARBINARY,
  `message` VARBINARY,
  PRIMARY KEY (messageKey)
) with (
  type = 'kafka010',
  topic = '<yourTopicName>',
  bootstrap.servers = '<yourServerAddress>'
);
```

> **Note**
> - When you create a Message Queue for Apache Kafka result table, you must specify `PRIMARY KEY (messageKey)` in plaintext mode.
> - Only Blink 2.2.6 and later versions support the display of metrics such as transactions per second (TPS) and requests per second (RPS) of Alibaba Cloud Message Queue for Apache Kafka or a self-managed Kafka database.

Parameters in the WITH clause

- General configurations

Parameter	Description	Required	Remarks
type	The Kafka version.	Yes	Valid values: Kafka08, Kafka09, Kafka010, and Kafka011. For more information, see Mapping between the values of the type parameter and Kafka versions.
topic	The name of the topic.	Yes	None.

- Required configurations
  - Required configurations for Kafka08

Parameter	Description
zookeeper.connect	The ZooKeeper URL.

  - Required configurations for Kafka09, Kafka010, and Kafka011

Parameter	Description
bootstrap.servers	The Kafka cluster address.

- Optional configurations
  - `consumer.id`
  - `socket.timeout.ms`
  - `fetch.message.max.bytes`
  - `num.consumer.fetchers`
  - `auto.commit.enable`
  - `auto.commit.interval.ms`
  - `queued.max.message.chunks`
  - `rebalance.max.retries`
  - `fetch.min.bytes`
  - `fetch.wait.max.ms`
  - `rebalance.backoff.ms`
  - `refresh.leader.backoff.ms`
  - `auto.offset.reset`
  - `consumer.timeout.ms`
  - `exclude.internal.topics`
  - `partition.assignment.strategy`
  - `client.id`
  - `zookeeper.session.timeout.ms`
  - `zookeeper.connection.timeout.ms`
  - `zookeeper.sync.time.ms`
  - `offsets.storage`
  - `offsets.channel.backoff.ms`
  - `offsets.channel.socket.timeout.ms`
  - `offsets.commit.max.retries`
  - `dual.commit.enabled`
  - `partition.assignment.strategy`
  - `socket.receive.buffer.bytes`
  - `fetch.min.bytes`

> **Note** For more information about optional configuration items, see the following official Kafka documentation:
>
> - Kafka09
> - Kafka010
> - Kafka011

Mapping between the values of the type parameter and Kafka versions

type	Kafka version
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2 and later

Example

```
create table datahub_input (
  id VARCHAR,
  nm VARCHAR
) with (
  type = 'datahub'
);

create table sink_kafka (
  messageKey VARBINARY,
  `message` VARBINARY,
  PRIMARY KEY (messageKey)
) with (
 type = 'kafka010',
 topic = '<yourTopicName>',
 bootstrap.servers = '<yourServerAddress>'
);

INSERT INTO
 sink_kafka
SELECT
cast(id as VARBINARY) as messageKey,
 cast(nm as VARBINARY) as `message`
FROM
 datahub_input;
```

5.6.3.14. Create a HybridDB for MySQL result table

This topic describes how to create a HybridDB for MySQL result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause used when you create such a result table.

> **Important**
>
> - HybridDB for MySQL is no longer available.
> - This topic applies only to Blink 1.4.5 and later.

Introduction to HybridDB for MySQL

HybridDB for MySQL (formerly known as PetaData) is a hybrid transaction/analytical processing (HTAP) relational database that supports both online transaction processing (OLTP) and online analytical processing (OLAP). HybridDB for MySQL uses the same data store for OLTP and OLAP. This prevents data replications during data analysis and significantly reduces data storage costs.

DDL syntax

In Realtime Compute for Apache Flink, you can use HybridDB for MySQL to store output data. The following code shows an example:

```
create table petadata_output(
 id INT,
 len INT,
 content VARCHAR,
 primary key(id,len)
) with (
 type='petaData',
 url='yourDatabaseURL',
 tableName='yourTableName',
 userName='yourDatabaseUserName',
 password='yourDatabasePassword'
);
```

> **Note**
>
> - In Realtime Compute for Apache Flink, each row of output data is converted to a line of SQL statement and then written and executed in the destination database.
> - The default value of the bufferSize parameter is 1000. If the number of data records reaches the value of this parameter, data is written into the result table. If you specify the batchSize parameter, you must also specify the bufferSize parameter. You can set the two parameters to the same value.
> - We recommend that you set the `batchSize` parameter to 4096. Do not set it to a large value.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to petaData.
url	The URL that is used to access the database.	Yes	Switch network type.

tableName	The name of the table.	Yes	None.
userName	The username that is used to access the database.	Yes	None.
password	The password that is used to access the database.	Yes	None.
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 3.
batchSize	The number of data records that can be written at a time.	No	Default value: 1000.
bufferSize	The maximum number of data records that can be stored in the buffer before deduplication is triggered.	No	None.
flushIntervalMs	The timeout period for writing data.	No	Unit: milliseconds. Default value: 3000. This value indicates that if the number of input data records does not reach the value specified by the bufferSize parameter within 3,000 milliseconds, all cached data is written into the result table.
ignoreDelete	Specifies whether to skip delete operations.	No	Default value: false.

5.6.3.15. Create an ApsaraDB RDS for SQL Server result table

This topic describes how to create an ApsaraDB RDS for SQL Server result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, data type mappings, and Java Database Connectivity (JDBC) parameters that are used when you create such a result table.

> **Important**
>
> - This topic applies only to Blink V3.2.0 and later.
> - Realtime Compute for Apache Flink cannot use ApsaraDB RDS for SQL Server as a data store.

DDL syntax

In Realtime Compute for Apache Flink, you can use ApsaraDB RDS for SQL Server to store output data. The following code shows an example:

```
create table ss_output(
 id INT,
 len INT,
 content VARCHAR,
 primary key(id,len)
) with (
 type='jdbc',
 url='jdbc:sqlserver://ip:port;database=****',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

> **Note**
>
> - In Realtime Compute for Apache Flink, each row of output data is converted to a line of SQL statement and then written and executed in the destination database. If you want to write multiple rows of data to the result table at the same time, you must add **?rewriteBatchedStatements=true** to the URL to improve system performance.
> - You can define an auto-increment primary key for an ApsaraDB RDS for SQL Server database. If you want to use the auto-increment primary key, do not declare the auto-increment field in the DDL statement. For example, if you use the ID field as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written into an ApsaraDB RDS for SQL Server database, a value is automatically filled for the auto-increment field.
> - If a DRDS result table has partitions, the shard key must be declared in **primary key()** of the DDL statement. Otherwise, you cannot write data into the partitioned table.
> - The fields that are declared in a DDL statement must include at least one non-primary key field. Otherwise, an error is returned.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
url	The JDBC URL of the database.	Yes	For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for MySQL instance.

tableName	The name of the table.	Yes	N/A.
username	The username that is used to access the database.	Yes	N/A.
password	The password that is used to access the database.	Yes	N/A.
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 10.
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	No	Default value: 10000. This value indicates that deduplication is triggered after the number of input data records reaches 10,000.
flushIntervalMs	The interval at which the cache is cleared.	No	Unit: milliseconds. Default value: 2000. This value indicates that if the number of input data records does not reach the value specified by the bufferSize parameter within 2,000 milliseconds, all cached data is written into the result table.
excludeUpdateColumns	Specifies whether to ignore the update of the specified field.	No	This parameter is optional. This parameter is empty by default, which indicates that the primary key field is ignored by default. When data with the same primary key is updated, the specified columns are not updated.
ignoreDelete	Specifies whether to ignore delete operations.	No	Default value: False.

Data type mappings

Data type of ApsaraDB RDS for SQL Server	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

JDBC parameters

Parameter	Description	Default value	Since version (JDBC driver)
useUnicode	Specifies whether to use the Unicode character set. This parameter must be set to True if you set the characterEncoding parameter to GB2312 or GBK.	False	1.1g

characterEncoding	The character encoding format, such as GB2312 or GBK. If useUnicode is set to true, you must specify a character encoding format.	False	1.1g
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 the database is read-only after it is automatically reconnected.	True	3.0.12
maxReconnects	The maximum number of reconnection attempts allowed. This parameter must be set if the autoReconnect parameter is set to True.	3	1.1
initialTimeout	The interval between two reconnection attempts. Unit: seconds. This parameter must be set if the autoReconnect parameter is set to True.	2	1.1
connectTimeout	The timeout period when you use a socket connection to access the database server. Unit: milliseconds.	Default value: 0. This value indicates that the connection never times out. This parameter is provided in JDK V1.4 and later.	3.0.1

socketTimeout	The timeout period for read and write operations on a socket connection. Unit: milliseconds.	Default value: 0. This value indicates that the read or write operation never times out.	3.0.1

Sample code

The following example describes how to create an ApsaraDB RDS for SQL Server result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'random'
);

CREATE TABLE rds_output(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id,len)
) WITH (
 type='jdbc',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTable>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);

INSERT INTO rds_output
SELECT id, len, content FROM source;
```

FAQ

- Q: When the output data of Realtime Compute for Apache Flink is written to an ApsaraDB RDS for SQL Server table, is the result table updated based on the primary key or is a new data record generated in the table?

  A: The processing method depends on whether the primary key is defined in the DDL statement.

  - If a primary key is defined in the DDL statement, the result table is updated by using `insert into on duplicate key update`. For a data record, if the primary key does not exist, the record is inserted into the table as a new row. If the value of the primary key field exists, the original row in the table is updated.
  - If no primary key is defined in the DDL statement, new data records are appended to the table by using `insert into`.

- Q: What do I need to pay attention to when I perform GROUP BY operations based on the unique index of an ApsaraDB RDS for SQL Server table?

  A: Pay attention to the following points:

  - Declare the unique index in the primary key of your job.

- An ApsaraDB RDS for SQL Server table has only one auto-increment primary key. Therefore, this auto-increment primary key cannot be declared as the primary key in a Realtime Compute for Apache Flink job.

5.6.3.16. Create an ApsaraDB for Redis result table

This topic describes how to create an ApsaraDB for Redis result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, the mappings between the field data types of ApsaraDB for Redis and Realtime Compute for Apache Flink, and the attribute fields used when you create an ApsaraDB for Redis result table.

> **Important**
> - This topic applies only to Blink V3.2.0 and later.
> - Realtime Compute for Apache Flink allows you to use self-managed Redis databases to store output data in result tables.

Introduction to ApsaraDB for Redis

ApsaraDB for Redis is a database service that is compatible with the protocols of the open source Redis system. It supports a hybrid of memory and hard disks for storage. ApsaraDB for Redis provides a hot standby architecture to ensure high availability. Based on the scalable cluster architecture, ApsaraDB for Redis can meet the business requirements for high throughputs, low-latency operations, and flexible configuration changes. Realtime Compute for Apache Flink allows you to store the output streaming data in ApsaraDB for Redis.

Syntax

You can use five data types when you write data to ApsaraDB for Redis result tables. To create an ApsaraDB for Redis result table, execute the following data definition language (DDL) statements:

- STRING type
  A DDL statement has two columns. The first column lists keys and the second column lists values. To insert data into an ApsaraDB for Redis result table, run the `set key value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'string',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data w
hen the retraction message is returned. The default value is false.
);
```

- LIST type
  A DDL statement has two columns. The first column lists keys and the second column lists values. To insert data into an ApsaraDB for Redis result table, run the `lpush key value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'list',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data w
hen the retraction message is returned. The default value is false.
);
```

- SET type
  A DDL statement has two columns. The first column lists keys and the second column lists values. To insert data into an ApsaraDB for Redis result table, run the `sadd key value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'set',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data w
hen the retraction message is returned. The default value is false.
);
```

- HASHMAP type
  A DDL statement has three columns. The first column lists keys, the second column lists hash keys, and the third column lists hash values. To insert data into an ApsaraDB for Redis result table, run the `hmset key hash_key hash_value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'hashmap',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data w
hen the retraction message is returned. The default value is false.
);
```

- SORTEDSET type
  A DDL statement has three columns. The first column lists keys, the second column lists

scores, and the third column lists values. To insert data into an ApsaraDB for Redis result table, run the `add key score value` command.

```
create table resik_output (
  a varchar,
  b double,  -- The data must be of the DOUBLE type.
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'sortedset',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data w
hen the retraction message is returned. The default value is false.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Valid value
type	The type of the result table.	Yes	Set the value to `redis`.
mode	The data type of the ApsaraDB for Redis result table.		Valid values: • string • list • set • hashmap • sortedset
host	The endpoint of the ApsaraDB for Redis database.		Example: `127.0.0.1`.
port	The port of the ApsaraDB for Redis database.	No	Default value: 6379.
dbNum	The sequence number of the ApsaraDB for Redis database.		Default value: 0.
ignoreDelete	Specifies whether to ignore the retraction message.		Valid values: true and false. Default value: false. If this parameter is set to false, the inserted data and the keys of the data are deleted when a retraction message is received.
password	The password that is used to access the ApsaraDB for Redis database.		This parameter is empty by default. This indicates that permission verification is not required.

clusterMode	Specifies whether the ApsaraDB for Redis database is in cluster mode.		Valid values: • true: cluster mode. • false: standalone mode. This is the default value. **Note** Only Blink 3.6.X and later support this parameter.

Field type mapping

The following table lists the data type mappings between ApsaraDB for Redis and Realtime Compute for Apache Flink. We recommend that you declare the mappings in DDL statements.

Data type of ApsaraDB for Redis	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR
SCORE	DOUBLE

Note The data of the SCORE type is added to the values of the SORTEDSET data type in ApsaraDB for Redis databases. You must manually set a score of the DOUBLE type for each sorted set value and sort the values based on their scores in ascending order.

Sample code

The following sample code shows how to create an ApsaraDB for Redis result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE random_stream (
  v VARCHAR,
  p VARCHAR) with (
    type = 'random'
);

create table resik_output (
  a VARCHAR,
  b VARCHAR,
  primary key(a)
) with (
  type = 'redis',
  mode = 'string',
  host = '<yourRedisHost>',
  password = '<yourRedisPassword>'
);

INSERT INTO resik_output
SELECT v, p
FROM random_stream;
```

5.6.3.17. Create an ApsaraDB for MongoDB result table

This topic describes how to create an ApsaraDB for MongoDB result table in Realtime Compute for Apache Flink. This parameter also describes the parameters in the WITH clause that is used when you create an ApsaraDB for MongoDB result table.

> **Important**
> - This topic applies to only Blink V3.2.2 and later.
> - You cannot update the primary key in an ApsaraDB for MongoDB result table. As a result, data that has the same primary key is inserted repeatedly into the table.

Syntax

In Realtime Compute for Apache Flink, you can use ApsaraDB for MongoDB to store output data. To create an ApsaraDB for MongoDB result table, you can use the following sample code:

```
CREATE TABLE mongodb_sink (
  `a`             VARCHAR
) WITH (
   type = 'mongodb',
   database = '<yourDatabaseName>',
   collection= '<yourCollectionName>',
   uri='mongodb://{<databaseAccount>}:{<atabasePassword>}@{host}:****?
replicaSet=mgset-1224****',
   keepAlive='true',
   maxConnectionIdleTime='20000',
   batchSize='2000'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the connector.	Yes	Set the value to mongodb.
database	The name of the ApsaraDB for MongoDB database.	Yes	None.
collection	The set of result table data.	Yes	None.
uri	The connection string of the ApsaraDB for MongoDB database.	Yes	None.

keepAlive	Specifies whether to maintain the persistent connection.	No	Default value: true.
maxConnectionIdleTime	The time-out duration of the connection.	No	The integer value. Unit: milliseconds. This parameter cannot be set to a negative value. Default value: 60000. If this parameter is set to 0, the connection does not time out.
batchSize	The number of data records that can be written at a time.	No	The integer value. Default value: 1024. The system sets the maximum number of data records that can be stored in the buffer. When the number of data records reaches the specified value of batchSize, the system triggers the data output. **Note** When the checkpoint time arrives, the data output is triggered even if the number of data records in the buffer does not reach the specified value of batchSize.

5.6.3.18. Create an AnalyticDB for MySQL V3.0 result table

This topic describes how to create an AnalyticDB for MySQL V3.0 result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause used when you create an AnalyticDB for MySQL V3.0 result table.

Important

- AnalyticDB for MySQL V3.0 result tables do not support the storage registration feature.
- This topic applies only to `Blink 3.3.0` and later.
- For more information about how to create an AnalyticDB for MySQL V2.0 result table, see Create an AnalyticDB for MySQL V2.0 result table.
- You can define an auto-increment primary key for an AnalyticDB for MySQL V3.0 database. If you want to use the auto-increment primary key, do not declare the auto-increment field in a DDL statement. For example, if you use ID as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written to the AnalyticDB for MySQL V3.0 database, the value for the auto-increment field is automatically filled.

DDL syntax

In Realtime Compute for Apache Flink, you can use AnalyticDB for MySQL V3.0 to store output data. The following sample code provides an example on how to create an AnalyticDB for MySQL V3.0 result table.

```
CREATE TABLE adb_output (
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id,len)
) WITH (
type='ADB30',
url='jdbc:mysql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>'
);
```

Principles

Realtime Compute for Apache Flink writes data to an AnalyticDB for MySQL V3.0 result table in two steps:

1. Converts each row of output data to a line of SQL statement.
2. Writes and executes the SQL statement in the destination database.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the connector.	Yes	Set the value to `ADB30`.
url	The Java Database Connectivity (JDBC) URL of the database.	Yes	The URL of the AnalyticDB for MySQL database.Example: `url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'`. **Note** • For more information about how to access an AnalyticDB for MySQL database, see Query a URL. • databaseName is the name of the AnalyticDB for MySQL database.
tableName	The name of the table in the database.	Yes	N/A.
username	The username that is used to access the AnalyticDB for MySQL database.	Yes	N/A.

password	The password that is used to access the AnalyticDB for MySQL database.	Yes	N/A.
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 3.
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	No	Default value: 1000. This value indicates that duplicates are removed when the number of input data records reaches 1,000. **Note** This parameter is valid only after you specify the primary key.
batchSize	The maximum number of data records that can be written at a time.	No	Default value: 1000. **Note** This parameter is valid only after you specify the primary key.
flushIntervalMs	The interval at which the cache is cleared.	No	Default value: 3000. Unit: milliseconds. This value indicates that all the cached data is written to the result table if the number of input data records does not reach the batchSize value within 3,000 milliseconds.
ignoreDelete	Specifies whether to ignore delete operations.	No	Default value: false. This value indicates that the delete operations are supported.
replaceMode	Specifies whether to use the REPLACE INTO statement to insert data into the table.	No	Valid values: • true: The REPLACE INTO statement is used to insert data into the table. This is the default value. • false: The INSERT INTO ON DUPLICATE KEY statement is used to insert data into the table. **Note** This parameter is valid only when the following conditions are met: • The Blink version is 3.X or later. • The AnalyticDB for MySQL version must be 3.1.3.5 or later.
excludeUpdateColumns	The columns that are not updated when data that has the same primary key is updated.	No	If you specify multiple columns that you do not want to update, separate the column names with commas (,). Example: `excludeUpdateColumns=column1,column2`. **Important** Make sure that the columns that you want to ignore are written in one line and cannot be wrapped. Otherwise, the setting does not take effect.

reserveMilliSecond	Specifies whether to reserve the millisecond component in a value of the TIMESTAMP data type.	No	Default value: false. This value indicates that the millisecond component is not reserved.

5.6.3.19. Create a custom result table

This topic describes how to create a custom result table in Realtime Compute for Apache Flink. Custom result tables can meet different data output requirements.

> **Important**
> - This topic applies only to Blink 1.4.5 and later.
> - This topic applies only to Realtime Compute for Apache Flink in exclusive mode.

Build a development environment

You can use one of the following methods to build a development environment for a custom result table:

- Use the development environment provided in examples.
  To accelerate the development of your services, Realtime Compute for Apache Flink provides the following examples of custom result tables:
  - Realtime Compute for Apache Flink V3.0
  - Realtime Compute for Apache Flink V2.0
  - Realtime Compute for Apache Flink V1.0

  > **Note** These examples provide development environments for specific versions. You do not need to build another development environment.

- Download a JAR package and build your own environment.

  > **Note** If the following dependencies are referenced in a Maven project, you must set the Scope parameter to `<scope>provided</scope>`.

  - Realtime Compute for Apache Flink V3.0

- JAR packages that you need to download
  - blink-connector-custom-blink-3.2.1
  - blink-connector-common-blink-3.2.1

  You must add the following information to the POM file to automatically download the `flink-table_2.11` JAR package.

```
<profiles>
 <profile>
  <id>allow-snapshots</id>
    <activation><activeByDefault>true</activeByDefault></activation>
  <repositories>
   <repository>
    <id>snapshots-repo</id>
    <url>https://oss.sonatype.org/content/repositories/snapshots</url>
    <releases><enabled>false</enabled></releases>
    <snapshots><enabled>true</enabled></snapshots>
   </repository>
  </repositories>
 </profile>
</profiles>
```

- Dependencies

```
 <dependencies>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-common</artifactId>
    <version>blink-3.2.1-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-custom</artifactId>
    <version>blink-3.2.1-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>flink-table_2.11</artifactId>
    <version>blink-3.2.1-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

- Realtime Compute for Apache Flink V2.0
  - JAR packages that you need to download
    - blink-connector-common-blink-2.2.4
    - blink-connector-custom-blink-2.2.4
    - blink-table-blink-2.2.4
    - flink-table_2.11-blink-2.2.4
    - flink-core-blink-2.2.4
  - Dependencies

```
<dependencies>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-table</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-core</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-common</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-custom</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

- Realtime Compute for Apache Flink V1.0
  - JAR packages that you need to download
    - blink-connector-common-blink-1.4
    - blink-connector-custom-blink-1.4
    - blink-table-blink-1.4
    - flink-core-blink-1.4
    - flink-streaming-scala_2.11-blink-1.4
  - Dependencies

```
      <dependencies>
        <dependency>
            <groupId>com.alibaba.blink</groupId>
            <artifactId>blink-connector-common</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba.blink</groupId>
            <artifactId>blink-connector-custom</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-scala_${scala.binary.version}
</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba.blink</groupId>
            <artifactId>blink-table</artifactId>
            <version>blink-1.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

API description

The class of a custom result table must inherit the CustomSinkBase base class of the custom sink plug-in and is implemented by using the following methods:

```
protected Map<String,String> userParamsMap;// userParamsMap is the key-value pair defined in the WITH clause of custom SQL statements. All keys are in lowercase letters.
protected Set<String> primaryKeys;// primaryKeys is the custom primary key field.
protected List<String> headerFields;// headerFields is the list of fields marked as header.
protected RowTypeInfo rowTypeInfo;// rowTypeInfo indicates the field type and name.
/**
 * The initialization method. This method is called when you create a table for the first time or when a failover occurs.
 *
 * @param taskNumber The serial number of the current node.
 * @ param numTasks The total number of sink nodes.
 * @throws IOException
 */
public abstract void open(int taskNumber,int numTasks) throws IOException;

/**
 * The close method that is used to release resources.
 *
 * @throws IOException
 */
public abstract void close() throws IOException;

/**
 * Insert a single row of data.
 *
 * @param row
 * @throws IOException
 */
public abstract void writeAddRecord(Row row) throws IOException;

/**
 * Delete a single row of data.
 *
 * @param row
 * @throws IOException
 */
public abstract void writeDeleteRecord(Row row) throws IOException;

/**
 * If you want to use this method to insert multiple rows of data at the same time, you must load all data cached in the threads to the downstream storage system. If you do not need to insert multiple rows of data at the same time, this method is not required.
 *
 * @throws IOException
 */
public abstract void sync() throws IOException;

/**
* Return the class name.
*/
public String getName();
```

Example of creating a custom ApsaraDB for Redis result table

Download Demo of Realtime Compute for Apache Flink V3.0. Go to the **blink_customersink_3x** directory, run the `mvn clean package` command, and then upload the JAR package **blink_customersink_3x/target/blink-customersink-3.x-1.0-SNAPSHOT-jar-with-dependencies.jar** that is compiled in the Realtime Compute for Apache Flink console. After you reference required resources, you must specify `type = 'custom'` for the custom sink plug-in, and specify the class for implementing the API.

> **Important** This example is only used as a reference for developing custom result tables. It is not suitable for production purposes.

```
create table in_table(
    kv varchar
)with(
    type = 'random'
);

create table out_table(
    `key` varchar,
    `value` varchar
)with(
    type = 'custom',
    class = 'com.alibaba.blink.customersink.RedisSink',
    -- 1. You can define more custom parameters. These parameters can be obtained by us
ing userParamsMap in the open function.
    -- 2. The keys for the parameters in the WITH clause are not case-sensitive. In Rea
ltime Compute for Apache Flink, the values of the parameter keys are processed as lower
case letters. We recommend that you declare keys in lowercase letters in the data defin
ition language (DDL) statements that reference the data store.
    host = 'r-uf****.redis.rds.aliyuncs.com',
    port = '6379',
    db = '0',
    batchsize = '10',
    password = '<yourHostPassword>'
);

insert into out_table
select
substring(kv,0,4) as `key`,
substring(kv,0,6) as `value`
from in_table;
```

The following table describes the parameters of the plug-in of the ApsaraDB for Redis sink.

Parameter	Description	Required	Remarks
host	The internal endpoint of the ApsaraDB for Redis instance.	Yes	None.
port	The number of the port that is used to access the ApsaraDB for Redis instance.	Yes	None.

password	The password that is used to access the ApsaraDB for Redis instance.	Yes	None.
db	The serial number of an ApsaraDB for Redis database.	No	Default value: 0. This value indicates db0.
batchsize	The number of data records that can be written at a time.	No	Default value: 1. This value indicates that multiple data records cannot be written at a time.

5.6.3.20. Create a Phoenix5 result table

This topic describes how to create a Phoenix5 result table in Realtime Compute for Apache Flink.

> **Important**
> - This topic applies only to Realtime Compute for Apache Flink in exclusive node.
> - This topic applies only to Blink 3.4.0 and later.
> - Only Phoenix 5.X is supported.
> - Phoenix is an HBase SQL service deployed on an ApsaraDB for HBase instance. You can use Phoenix only after you activate this service in ApsaraDB for HBase instances.

DDL syntax

In Realtime Compute for Apache Flink, you can use Phoenix5 to store output data. The following code shows an example:

```
create table US_POPULATION_SINK (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY)--- The primary key. This field is required.
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to `PHOENIX5`.

serverUrl	The URL of the Phoenix5 query server. • If Phoenix5 is created in a cluster, the value of this parameter is the URL of Server Load Balancer (SLB). • If Phoenix5 is created on a single server, the value of this parameter is the URL of the server.	Yes	You must enable the HBase SQL service in an ApsaraDB for HBase instance. The value of the serverUrl parameter is in the **http://host:port** format. In the format: • host: indicates the domain name of the Phoenix5 service. • port: indicates the port number of the Phoenix5 service. Set the value to 8765.
tableName	The name of the Phoenix5 table.	Yes	The name of the Phoenix5 table is in the SchemaName.TableName format. In the format: • SchemaName: indicates the schema name, which can be empty. This means that the schema name is not used and only the table name is used. In this case, the default schema of the database is used. • TableName: the name of the table.

Sample code

The following sample code shows how to create a Phoenix5 result table in a Realtime Compute for Apache Flink job.

```
create table `source` (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar
) WITH (
   type = 'random'
);

create table sink (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar,
  primary key (id)
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);

INSERT INTO sink
  SELECT  `id` ,`name` , `age` ,`birthday`
FROM `source`;
```

5.6.3.21. Create an AnalyticDB for PostgreSQL result table

This topic describes how to create an AnalyticDB for PostgreSQL result table. This topic also describes the parameters in the WITH clause and data type mappings used when you create an AnalyticDB for PostgreSQL result table.

> **Important** This topic applies to only Blink 3.6.0 and later.

Principles

Realtime Compute for Apache Flink writes data to an AnalyticDB for PostgreSQL result table in two steps:

1. Converts each row of output data to a line of SQL statement.
2. Writes and executes the SQL statement in the destination database.

DDL syntax

In Realtime Compute for Apache Flink, you can use AnalyticDB for PostgreSQL to store output data. The following sample code shows how to create an AnalyticDB for PostgreSQL result table.

```
create table rds_output(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY(id)
) with (
  type='adbpg',
  url='jdbc:postgresql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to `adbpg`.
url	The Java Database Connectivity (JDBC) URL of the database.	Yes	The JDBC URL used to access the AnalyticDB for PostgreSQL database. The format is **'jdbc:postgresql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>'**. • yourNetworkAddress: the internal IP address. • PortId: the port that is used to log on to the database. • yourDatabaseName: the name of the database. For example, you can set yourDatabaseName to **url='jdbc:postgresql://gp-xxxxxx.gpdb.cn-chengdu.rds.aliyuncs.com:3432/postgres'**.
tableName	The name of the table.	Yes	N/A
username	The account that is used to access the AnalyticDB for PostgreSQL database.	Yes	N/A
password	The password that is used to access the AnalyticDB for PostgreSQL database.	Yes	N/A
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 3.

useCopy	Specifies whether to use the copy API to write data.	No	Valid values: • 0: indicates that the INSERT statement is executed to write data to the AnalyticDB for PostgreSQL database. This is the default value. • 1: indicates that the copy API is used to write data to the AnalyticDB for PostgreSQL database.
batchSize	The number of data records that can be written at a time.	No	Default value: 5000.
exceptionMode	The policy that is used to handle exceptions during data writing.	No	Valid values: • ignore: The system ignores the data that is written when exceptions occur. This is the default value. • strict: If an exception occurs during data writing, an error message appears on the Failover tab.
conflictMode	The policy that is used to handle primary key conflicts or unique index conflicts.	No	Valid values: • ignore: Primary key conflicts are ignored and the existing data is retained. This is the default value. • strict: If a primary key conflict occurs, an error message appears on the Failover tab. • update: If a primary key conflict occurs, data is updated.
targetSchema	The name of the schema.	No	Default value: public.
connectionMaxActive	The maximum number of connections allowed for a single task.	No	Configure this parameter based on the actual number of parallel tasks and the maximum number of connections allowed to the destination database.

Data type mapping

The following table lists the mappings between the field data types of AnalyticDB for PostgreSQL and Realtime Compute for Apache Flink.

Data type of AnalyticDB for PostgreSQL	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
SMALLINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
DOUBLE PRECISION	DOUBLE
TEXT	VARCHAR
TIMESTAMP	DATETIME

DATE	DATE
REAL	FLOAT
DOUBLE PRECISION	DECIMAL
TIME	TIME
TIMESTAMP	TIMESTAMP

5.6.3.22. Create an InfluxDB result table

This topic describes how to create an InfluxDB result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause and data type mapping involved when you create an InfluxDB result table.

> **Important**
> - InfluxDB does not support the storage registration feature.
> - This topic applies only to Blink 3.5.0-hotfix and later.

DDL syntax

In Realtime Compute for Apache Flink, you can use InfluxDB to store output data. The following code shows an example:

```
create table stream_test_influxdb(
    `metric` varchar,
    `timestamp` BIGINT,
    `tag_value1` varchar,
    `field_fieldValue1` Double
)with(
    type = 'influxdb',
    endpoint = 'http://service.cn.influxdb.aliyuncs.com:****',
    database = '<yourDatabaseName>',
    batchPutsize = '1',
    username = '<yourDatabaseUserName>',
    password = '<yourDatabasePassword>'
);
```

Default format for the created table:

- Column 0: metric (VARCHAR). This column is required.
- Column 1: timestamp (BIGINT). This column is required. Unit: milliseconds.
- Column 2: tag_value1 (VARCHAR). This column is required. You must enter at least one value in this column.
- Column 3: field_fieldValue1 (DOUBLE). This column is required. You must enter at least one value in this column.
  To specify multiple field_fieldValue values, use the following format:

```
field_fieldValue1 <Data type>,
field_fieldValue2 <Data type>,
...
field_fieldValueN <Data type>
```

The following code shows an example:

```
field_fieldValue1 Double,
field_fieldValue2 INTEGER,
...
field_fieldValueNINTEGER
```

> **Note** An InfluxDB result table can contain only **metric**, **timestamp**, **tag_***, and **field_***.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to InfluxDB.
endpoint	The endpoint of the InfluxDB database.	Yes	The endpoint of an InfluxDB database is the VPC endpoint of the InfluxDB database. For example, you can set this parameter to https://localhost:3242 or http://localhost:8086. Endpoints support HTTP and HTTPS.
database	The name of the InfluxDB database.	Yes	For example, you can set this parameter to db-blink or blink.
batchPutSize	The number of data records that are submitted at a time.	No	Default value: 500.
username	The username that is used to access the InfluxDB database.	Yes	You must have the write permission on the InfluxDB database.
password	The password that is used to access the InfluxDB database.	Yes	Default value: 0.
retentionPolicy	The retention policy.	No	If this parameter is empty, the default retention policy is used for each database.

Field type mapping

InfluxDB data type	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
INT	INT

BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR

5.6.4. Create a dimension table

5.6.4.1. Overview

This topic describes how to use the standard CREATE TABLE statement to create a dimension table in Realtime Compute for Apache Flink. To use the standard data definition language (DDL) statement to create a dimension table, add PERIOD FOR SYSTEM_TIME to the statement to define the change period of the dimension table.

Example

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME  --Define the change period of the dimension table. In Realtime Compute for Apache Flink V3.X and later, you do not need to declare PERIOD FOR SYSTEM_TIME. You need to declare only FOR SYSTEM_TIME AS OF PROCTIME() when you join a dimension table with another table.
) with (
  type = 'RDS',
  ...
);
```

> **Note**
> - You must specify a primary key for each dimension table. When you join a dimension table with another table, the ON clause must contain the equivalent (=) conditions for all the primary key fields.
> - You can execute only the `INNER JOIN` or the `LEFT JOIN` statement to join a source table and a dimension table.
> - The unique key of the dimension table must be the same as that of the database table. If they are not the same, you may encounter the following issues:
>   - Reading data from the dimension table slows down.
>   - When you join the dimension table with another table, the join operation starts from the first data record. In the Realtime Compute for Apache Flink job, multiple data records that have the same key are sequentially updated in the database. This may cause errors in the join result.

INDEX syntax

> **Note** We recommend that you use the `INDEX` syntax in Realtime Compute for Apache Flink V2.2.7 and later.

In Realtime Compute for Apache Flink versions that are earlier than V2.2, you must declare the `PRIMARY KEY` when you create a dimension table. In this case, you can perform only one-to-one table joins. The `INDEX` syntax is introduced to meet the requirements for one-to-many table joins. For dimension tables that do not support the `ALL` cache policy, you can use `INDEX LOOKUP` to meet the requirements for one-to-many table joins.

```
CREATE TABLE Persons (
    ID bigint,
    LastName varchar,
    FirstName varchar,
    Nick varchar,
    Age int,
    [UNIQUE] INDEX(LastName,FirstName,Nick), --Define the index. You do not need to spe
cify the index type, such as fulltext or clustered.
    PERIOD FOR SYSTEM_TIME
) with (
  type='RDS',
  ...
);
```

`UNIQUE INDEX` represents a **one-to-one** table join. `INDEX` represents a **one-to-many** table join.

Note

- `UNIQUE CONSTRAINT` (`UNIQUE KEY`) is supported in Realtime Compute for Apache Flink V2.2.7 and later. In Realtime Compute for Apache Flink versions that are earlier than V2.2.7, you can define the index by using `PRIMARY KEY` .
- The engine preferentially uses `UNIQUE INDEX` when it generates an execution plan. If INDEX is used in the DDL statement and the JOIN equivalent (=) conditions include both `UNIQUE` and `NON-UNIQUE INDEX` , the system preferentially uses `UNIQUE INDEX` to search for data in the right table.
- The dimension table types, such as ApsaraDB RDS dimension table and MaxCompute dimension table, supports one-to-many table joins.
- In one-to-many table joins, you can use the `maxJoinRows` parameter to specify the maximum number of associated rows in the right table for each row in the left table. The default value is 1024. If one row is associated with an excessively large number of rows, the performance of stream processing tasks may be compromised. If this occurs, you can increase the cache size. You can use the `cacheSize` parameter to limit the number of keys in the left table.
- The INDEX syntax cannot be used to perform one-to-many table joins on Tablestore and Hologres dimension tables.

Differences among dimension tables, source tables, and result tables

Item	Source Table	Result table	Dimension table
Trigger computing	Supported	Not supported	Not supported
Read data	Supported (You can directly read data from source tables.)	Not supported	Supported (You can read data from dimension tables only by joining a dimension table with a source table.)
Write data	Not supported	Supported	Not supported
Cache data	Not supported	Not supported	Supported

5.6.4.2. Create a Hologres dimension table

This topic describes how to create a Hologres dimension table. This topic also describes the parameters in the WITH clause, cache parameters, and data type mappings used when you create a Hologres dimension table.

Important

- This topic applies only to Blink 3.6.0 and later. If your Blink version is earlier than 3.6.0, you can submit a ticket to obtain the required Java Archive (JAR) files for the upgrade.
- We recommend that you use Hologres 0.7 or later.

What is Hologres?

Hologres is compatible with the PostgreSQL protocol and closely connected to the big data ecosystem. Hologres supports real-time analysis and processing of petabytes of data with high concurrency and low latency. This allows you to use existing Business Intelligence (BI) tools to perform multidimensional analysis and business exploration.

Limits

- We recommend that you use row-oriented storage to create a Hologres dimension table. Column-oriented storage consumes a large number of performance overheads for point queries.
  When you use row-oriented storage to create a Hologres dimension table, you must set the primary key to clustering key. The following statements show an example:

```
begin;
create table test(a int primary key, b text, c text, d float8, e int8);
call set_table_property('test', 'orientation', 'row');
call set_table_property('test', 'clustering_key', 'a');
commit;
```

- When you join a Hologres dimension table with another table, you must specify all the fields in the primary key of the dimension table in the ON clause.
- Hologres does not support a one-to-many mapping between input and output when you use Hologres connectors to join dimension tables.
- You cannot read data from Hologres partitioned tables.

DDL syntax

In Realtime Compute for Apache Flink, you can use a Hologres table as a dimension table. The following code shows an example.

```
CREATE TABLE hologres_dim_table(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME  --Define the change period of the dimension table.
) WITH (
  type='hologres',
  endpoint='...',
  dbname='...',
  tablename='...',
  username='...',
  password='...'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the database.	Yes	Set the value to hologres.
endpoint	The endpoint of the Hologres instance.	Yes	N/A.

tablename	The name of the table. **Note** If the public schema is not used, you must set tableName to **schema.tableName**.	Yes	N/A.
dbname	The name of the database.	Yes	N/A.
username	The username that is used to access the database.	Yes	N/A.
password	The password that is used to access the database.	Yes	N/A.

Cache parameters

Parameter	Description	Required	Remarks
cache	The cache policy.	No	The following cache policies are supported: • None: indicates that data is not cached. This is the default value. • LRU: indicates that only the specified data in the dimension table is cached. Each time the system receives a data record, the system searches the cache. If the system does not find the record in the cache, the system searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters.
cacheSize	The maximum number of rows of data records that can be cached.	No	This parameter is available when you set the **cache** parameter to LRU. Default value: 10000.
cacheTTLMs	The timeout period of the cache. Unit: milliseconds.	No	You can set this parameter when you set the **cache** parameter to LRU. By default, the cached data does not expire.

partitionedJoin	Specifies whether to use the joinKey for partitioning.	No	Valid values: • false: The joinKey is not used for partitioning. • true: The joinKey is used for partitioning. Data is distributed to JOIN nodes to improve the cache hit rate.
async	Specifies whether to read data in asynchronous mode.	No	**Valid values:** • false: Data is read synchronously. This is the default value. • true: Data is read asynchronously.

Data type mappings

Data type of Hologres	Data type of Realtime Compute for Apache Flink
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

Sample code

In Realtime Compute for Apache Flink, you can use a Hologres table as a dimension table. The following code shows an example.

```
create table randomSource (a int, b VARCHAR, c VARCHAR) with (type = 'random');

create table test (
  a int,
  b VARCHAR,
  c VARCHAR,
  PRIMARY KEY (a, b), PERIOD FOR SYSTEM_TIME
) with (
  type = 'hologres',
  ...
);

create table print_sink (
  a int,
  b VARCHAR
) with (
  type = 'print',
  `ignoreWrite` = 'false'
);

insert into print_sink
select randomSource.a, test.b from randomSource
LEFT JOIN test FOR SYSTEM_TIME AS OF PROCTIME()
on randomSource.a = test.a and randomSource.b = test.b;
```

5.6.4.3. Create a Tablestore dimension table

This topic describes how to create a Tablestore dimension table in Realtime Compute for Apache Flink.

> **Important** This topic applies only to Blink 1.4.5 and later.

Introduction to Tablestore

Tablestore is a distributed NoSQL database service that is built on the Apsara distributed operating system of Alibaba Cloud. Tablestore adopts sharding and load balancing technologies to scale out services and handle concurrent transactions. You can use Tablestore to store and query large amounts of structured data in real time.

Example

In Realtime Compute for Apache Flink, you can use a Tablestore table as a dimension table. The following code shows an example:

```
CREATE TABLE ots_dim_table (
 id int,
 len int,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME--Define the change period of the dimension table.
) WITH (
 type='ots',
 endPoint='<yourEndpoint>',
 instanceName='<yourInstanceName>',
 tableName='<yourTableName>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>'
);
```

Note

- When you declare a dimension table, you must specify a primary key.
- When you join a dimension table with another table, the ON clause must contain the equivalent (=) conditions for all the primary key fields.
- The primary key of a Tablestore table is the row key of the table.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the dimension table.	Set the value to `ots`.
endPoint	The endpoint of the Tablestore instance.	Enter the VPC endpoint if the instance is deployed in a VPC.
instanceName	The name of the Tablestore instance.	None.
tableName	The name of the Tablestore dimension table.	None.
accessId	The AccessKey ID that is used to access Tablestore.	None.
accessKey	The AccessKey secret that is used to access Tablestore.	None.

Parameters in the CACHE clause

Parameter	Description	Remarks

cache	The cache policy.	Valid values: • None: indicates that data is not cached. This is the default cache policy. • LRU: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If you use this cache policy, you must specify the cacheSize and cacheTTLMs parameters.
cacheSize	The maximum number of rows that can be cached.	You can set this parameter when the cache parameter is set to LRU. Default value: 10000.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	You can set this parameter when the cache parameter is set to LRU.

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 -- The identifier of a dimension table.
)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 w
hen you perform a JOIN operation on the dimension table.
ON t.name = w.name;
```

For more information about the syntax for dimension tables, see JOIN statements for dimension tables.

5.6.4.4. Create an ApsaraDB RDS for MySQL dimension table

This topic describes how to create an ApsaraDB RDS for MySQL dimension table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, cache parameters, and data type mapping used when you create an ApsaraDB RDS for MySQL dimension table.

ApsaraDB RDS for MySQL

ApsaraDB RDS for MySQL is developed based on a branch of MySQL source code and provides excellent performance. ApsaraDB RDS for MySQL is a tried and tested solution that handled large volumes of concurrent traffic during Double 11. ApsaraDB RDS for MySQL provides basic features such as whitelist configuration, backup and restoration, Transparent Data Encryption (TDE), data migration, and management of instances, accounts, and databases. For more information about ApsaraDB RDS for MySQL, see Overview.

Limits

Realtime Compute for Apache Flink does not allow you to use ApsaraDB RDS for MySQL V8.0 by using the storage registration method. To use ApsaraDB RDS for MySQL V8.0, we recommend that you configure a plaintext AccessKey pair. For more information about the storage registration method, see Overview.

DDL syntax

The following sample code shows how to create an ApsaraDB RDS for MySQL dimension table:

```
CREATE TABLE rds_dim_table(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME --Define the change period of the dimension table.
) with (
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

> **Note**
>
> You must specify a primary key when you declare a dimension table. When you join a dimension table with another table, the ON condition must contain equivalent conditions that include all primary keys. The primary key of an ApsaraDB RDS for MySQL or Distributed Relational Database Service (DRDS) database can be defined as the primary key or unique index column of an ApsaraDB RDS for MySQL or DRDS dimension table.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `rds`.

url	The Java Database Connectivity (JDBC) URL of the database.	Yes	Set the value in the `jdbc:mysql://<Internal endpoint>/<databaseName>` format. Replace **databaseName** with the name of your database. For more information about the internal endpoint, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for MySQL instance.
tableName	The name of the table.	Yes	N/A.
userName	The username that is used to access the ApsaraDB RDS database.	Yes	N/A.
password	The password that is used to access the ApsaraDB RDS database.	Yes	N/A.
maxRetryTimes	The maximum number of connection attempts.	No	Default value: 10.

Cache parameters

Parameter	Description	Required	Remarks

cache	The policy that is used to cache data.	No	Valid values: • None: indicates that data is not cached. This is the default cache policy. • LRU: indicates that only the specified data in the dimension table is cached. Each time the system receives a data record, the system searches the cache. If the system does not find the record in the cache, the system searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • ALL: indicates that all the data in the dimension table is cached. Before the system runs a job, the system loads all data in the dimension table to the cache. This way, the cache is searched for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data in a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. The source table and dimension table cannot be associated based on the ON clause. If you use this cache policy, you must configure the cacheTTLMs and cacheReloadTimeBlackList parameters. **Note** If you set the cache parameter to ALL, you must increase the memory of the node for joining tables because the system asynchronously loads data from the dimension table. The increased memory size is twice the memory size of the remote table.
cacheSize	The maximum number of rows of data records that can be cached.	No	This parameter is available only if you set the cache parameter to `LRU`. Default value: 10000.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	If the cache parameter is set to `LRU`, the cacheTTLMs parameter specifies the time allowed before cache entries expire. Cache entries do not expire by default. If the cache parameter is set to `ALL`, the cacheTTLMs parameter specifies the interval at which the cache is loaded. The cache is not reloaded by default.

cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter takes effect when the cache parameter is set to `ALL`. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	No	This parameter is empty by default. For example, you can set this parameter to `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00`. Multiple time periods are separated by commas (,). The start time and end time of each time period are separated by a hyphen and a greater-than sign (->).
maxJoinRows	The maximum number of results that are returned each time a data record in the primary table is queried and matched with data records in the dimension table.	No	Default value: 1024. If you can estimate that a data record in the primary table corresponds to a maximum of n data records in the dimension table, you can set the **maxJoinRows** parameter to n to ensure efficient matching in Realtime Compute for Apache Flink. **Note** When you join a dimension table with another table, this parameter specifies the maximum number of results that can be returned after a data record in the primary table is matched with data records in the dimension table.

Sample code

The following sample code shows how to create an ApsaraDB RDS dimension table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2017-07-21 00:00:00'
);

create table phoneNumber(
name VARCHAR,
phoneNumber BIGINT,
primary key(name),
PERIOD FOR SYSTEM_TIME--Define the change period of the dimension table.
)WITH(
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);

CREATE table result_infor(
id BIGINT,
phoneNumber BIGINT,
name VARCHAR
)WITH(
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);

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 w
hen you perform a JOIN operation on the dimension table.
ON t.name = w.name;
```

For more information about the detailed syntax of the JOIN statements for a dimension table, see JOIN statements for dimension tables.

Data type mapping

Data type of ApsaraDB RDS	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

5.6.4.5. Create an ApsaraDB for HBase dimension table

This topic describes how to create an ApsaraDB for HBase dimension table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH and CACHE clauses used when you create such a dimension table.

> **Important**
> - Blink versions earlier than Blink 3.3.0 support only HBase Standard Edition.
> - Blink 3.3.0 and later versions support HBase Standard Edition and HBase Enhanced Edition.
> - Blink 3.5.0 and later versions support switchover between primary and secondary ApsaraDB for HBase databases for data writing.
> - For more information about the JOIN syntax of an ApsaraDB for HBase dimension table, see JOIN statements for dimension tables.
> - ApsaraDB for HBase dimension tables in Realtime Compute for Apache Flink do not support user-created open source HBase.
> - Only one primary key is allowed in an ApsaraDB for HBase dimension table.

DDL syntax

- HBase Standard Edition

```
CREATE TABLE hbase (
  `key` varchar,
  `name` varchar,
  PRIMARY KEY (`key`), -- The rowkey field of the ApsaraDB for HBase dimension table.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
  TYPE = 'cloudhbase',
  zkQuorum = '<yourzkQuorum>',
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
```

- HBase Enhanced Edition

```
CREATE TABLE hbase (
  `key` varchar,
  `name` varchar,
  PRIMARY KEY (`key`), -- The rowkey field of the ApsaraDB for HBase dimension table.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
  TYPE = 'cloudhbase',
  endPoint = '<host:port>',-- The Java API URL that is used to access the Enhanced Edition of an ApsaraDB for HBase database.
  userName  = 'root', -- The username that is used to access an ApsaraDB for HBase database.
  password = 'root', -- The password that is used to access an ApsaraDB for HBase database.
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
```

- HBase Enhanced Edition for Blink 3.5.0 and later

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', -- The Java API URL that is used to access the Enhanced E
dition of an ApsaraDB for HBase database.
  userName  = 'root', -- The username that is used to access an ApsaraDB for HBase da
tabase.
  password = 'root', -- The password that is used to access an ApsaraDB for HBase dat
abase.
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

- Blink 3.5.0 and later versions support switchover between primary and secondary ApsaraDB for HBase databases for data writing.

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', -- The URL that is used to access ApsaraDB for HBase data
bases in high availability (HA) mode.
 haClusterID = 'ha-xxx', -- The instance ID of ApsaraDB for HBase databases in HA mod
e.
  userName  = 'root', -- The username that is used to access an ApsaraDB for HBase da
tabase.
  password = 'root', -- The password that is used to access an ApsaraDB for HBase dat
abase.
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

> **Note**
> - When you declare a dimension table, you must specify a primary key.
> - When you join a dimension table with another dimension table, the ON condition must contain equivalent conditions that include all primary keys. The primary key in the preceding examples is **row_key**.
> - The connection parameters in HBase Standard Edition and HBase Enhanced Edition are different.
>   - HBase Standard Edition: `zkQuorum`.
>   - HBase Enhanced Edition: `endPoint`.
>   - HBase Standard Edition and HBase Enhanced Edition for Blink 3.5.0 and later: `zkQuorum`.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `cloudhbase`.
zkQuorum	The ZooKeeper address configured for the ApsaraDB for HBase cluster. The address is a list of hosts separated by commas (,).	Yes	You can view the configuration related to **hbase.zookeeper.quorum** in the **hbase-site.xml** file. **Note** This parameter takes effect only in HBase Standard Edition.
zkNodeParent	The path of the cluster configured on the ZooKeeper servers.	No	You can view the configuration related to **hbase.zookeeper.quorum** in the **hbase-site.xml** file. **Note** This parameter takes effect only in HBase Standard Edition.
endPoint	The name of the region where your ApsaraDB for HBase instance is deployed.	Yes	You can obtain the value of this parameter from the console of your ApsaraDB for HBase instance. **Note** This parameter takes effect only in HBase Enhanced Edition.
userName	The username that is used to log on to the ApsaraDB for HBase database.	No	**Note** This parameter takes effect only in HBase Enhanced Edition.
password	The password that is used to log on to the ApsaraDB for HBase database.	No	**Note** This parameter takes effect only in HBase Enhanced Edition.

tableName	The name of the ApsaraDB for HBase dimension table.	Yes	None.
columnFamily	The column family name.	Yes	Only the same column family can be inserted.
maxRetryTimes	The maximum number of retries for writing data into the table.	No	Default value: 10.
partitionedJoin	Specifies whether to use joinKey for partitioning.	No	Default value: False. If you set this parameter to True, joinKey is used for partitioning. Data is delivered to each node for joining, which increases the cache hit ratio.
shuffleEmptyKey	Specifies whether to randomly send upstream empty keys to downstream nodes.	No	Default value: True. Valid values: • True: If multiple empty keys exist in the upstream, Realtime Compute for Apache Flink randomly sends all the empty keys to each JOIN node. • False: If multiple empty keys exist in the upstream, Realtime Compute for Apache Flink sends all the empty keys to a single JOIN node. **Note** This parameter can be used only after the **partitionedJoin** parameter takes effect.

Parameters in the CACHE clause

Parameter	Description	Required	Remarks

cache	The policy for caching data.	No	Valid values: • None (default value): indicates that no data is cached. • LRU: indicates that partial data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • ALL: indicates that all data in the dimension table is cached. Before a Realtime Compute for Apache Flink job starts to run, Realtime Compute for Apache Flink loads all data in the dimension table to the cache, and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If this cache policy is used, you must configure the cacheTTLMs and cacheReloadTimeBlackList parameters. **Note** If the cache parameter is set to ALL, the memory of the node for joining tables must be increased because Realtime Compute for Apache Flink asynchronously loads data from the dimension table. The increased memory size is two times that of the remote table.
cacheSize	The maximum number of data records that can be cached. Unit: lines.	No	You can set this parameter when the cache parameter is set to `LRU`. Default value: 10000.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	If the cache parameter is set to `LRU`, the cacheTTLMs parameter specifies the time before cache entries expire. Cache entries do not expire by default. If the cache parameter is set to `ALL`, the cacheTTLMs parameter specifies the interval at which the cache is loaded. The cache is not reloaded by default.

cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter is used when the cache parameter is set to `ALL`. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for massive online promotional events such as Double 11.	No	This parameter is empty by default. Custom input format: `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00` Separate multiple time periods with commas (,). Separate the start time and end time of each time period with a hyphen and a greater-than sign (->).
cacheScanLimit	The maximum number of rows returned by the server to the client for each remote procedure call (RPC) when the server reads full ApsaraDB for HBase data. This parameter is used when the cache parameter is set to `ALL`. .	No	Default value: 100.

Sample code

The following sample code demonstrates how to create an ApsaraDB for HBase dimension table in a Realtime Compute for Apache Flink job.

```
create table source (
  id   TINYINT,
  name BIGINT
) with (
  type = 'random'
);

create table dim (
  id   TINYINT,
  score BIGINT
  primary key(id),
  PERIOD FOR SYSTEM_TIME
)with(
  type = 'cloudhbase',
  zkQuorum = '<yourzkQuorum>',
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);

CREATE table result_infor(
  id BIGINT,
  score BIGINT
)with(
  type='rds'
);

INSERT INTO result_infor
SELECT
  t.id,
  w.score
FROM source as t
JOIN dim FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.id = w.id;
```

5.6.4.6. Create a MaxCompute dimension table

This topic describes how to create a MaxCompute dimension table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, cache parameters, and data type mappings used when you create a MaxCompute dimension table.

> **Important**
>
> - Blink 2.1.1 and later versions support MaxCompute dimension tables.
> - For more information about the query syntax of a dimension table, see JOIN statements for dimension tables.
> - To use a MaxCompute dimension table, you must grant the read permissions to the account used to access MaxCompute.

DDL syntax

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) WITH (
  type = 'odps',
  endPoint = '<YourEndPoint>',
  project = '<YourProjectName>',
  tableName = '<YourtableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****',
  cache = 'ALL'
);
```

> **Note**
>
> - When you declare a dimension table, you must specify a primary key. The primary key of a MaxCompute dimension table must be unique. Duplicate primary keys are deleted.
> - When you join a dimension table with another table, the ON condition must contain equality conditions that include all primary keys.
> - partition is a keyword and must be commented with backticks (`), for example, `'partition'`.
> - If the dimension table is a partitioned table, Realtime Compute for Apache Flink does not write partition key columns to the DDL statement.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `odps`.
endPoint	The endpoint of MaxCompute.	Yes	For more information, see Endpoints.
tunnelEndpoint	The endpoint of MaxCompute Tunnel.	Yes	For more information, see Endpoints. **Note** This parameter is required if MaxCompute is deployed in a virtual private cloud (VPC).

project	The name of the MaxCompute project.	Yes	N/A.
tableName	The name of the table.	Yes	N/A.
accessId	The AccessKey ID that is used to access MaxCompute.	Yes	N/A.
accessKey	The AccessKey secret that is used to access MaxCompute.	Yes	N/A.

partition	The name of a partition.	No	• Static partitions ◦ A MaxCompute table that has only one partition For example, if only one partition key column `ds` exists, `` `partition` = 'ds=20180905' `` indicates that data in the `ds=20180905` partition is read. ◦ A MaxCompute table that has multiple partitions For example, if two partition key columns `ds` and `hh` exist, `` `partition`='ds=20180905,hh=*' `` indicates that data in the `ds=20180905` partition is read. **Note** When you filter partitions, you must declare the values of all partitions. In the preceding example, if you declare only `` `partition` = 'ds=20180905' ``, no partition data is read. • Dynamic partitions ◦ Blink 2.2.0 and later support `'partition' = 'max_pt()'`. This setting indicates that the partition ranked first in alphabetical order among all partitions is loaded each time the system loads data of partitions. ◦ Blink 3.2.2 and later support `'partition' = 'max_pt_with_done()'`. This setting indicates that the partition with the file name extension `.done` and ranked first in alphabetical order among all partitions is loaded each time the system loads data of partitions.
maxRowCount	The maximum number of rows that Realtime Compute for Apache Flink can load from a table.	No	Default value: 100000. **Note** If your data contains more than 100,000 rows, you must configure this parameter. We recommend that you set this parameter to a greater value than the actual number of rows to be loaded.

Cache parameters

Parameter	Description	Remarks
cache	The cache policy.	You must set the cache parameter to `ALL` for a MaxCompute dimension table and explicitly declare the setting in the DDL statement. ALL: indicates that all data in the dimension table is cached. Before the system runs a job, the system loads all data in the dimension table to the cache. This way, the cache is searched for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data in a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. The source table and dimension table cannot be associated based on the ON clause. If you set this parameter to ALL, you must configure the cacheTTLMs and cacheReloadTimeBlackList parameters. **Note** • If the cache parameter is set to ALL, you must increase the memory of the join node because the system asynchronously loads data of the dimension table. We recommend that you increase the size of the memory at least four times the amount of data in the remote table. The size of the memory is related to the MaxCompute storage compression algorithm. • If a job exception occurs due to frequent garbage collection when you use a super-large MaxCompute dimension table, and this issue persists even if you increase the memory of the join node, we recommend that: ◦ For Blink 3.6.0 and later, set the **partitionedJoin** parameter to true to enable partitionedJoin optimization. ◦ Use a dimension table with key-value pairs for which you can set the cache parameter to LRU, such as an ApsaraDB for HBase dimension table.
cacheSize	The maximum number of data records that can be cached.	You can configure the cacheSize parameter based on your business requirements. By default, data of 100,000 rows in a MaxCompute dimension table can be cached.

cacheTTLMs	The cache timeout period.	Unit: milliseconds. If you set the cache parameter to `ALL`, the timeout period specifies the interval at which Realtime Compute for Apache Flink refreshes the cache. The cache is not refreshed by default.
cacheReloadTimeBlackList	The periods of time during which cache is not refreshed. This parameter takes effect when the cache parameter is set to ALL. The cache is not refreshed during the periods of time that you specify for this parameter. This parameter is suitable for large-scale online promotional events such as Double 11.	This parameter is empty by default. The following example shows the format of the values: `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00`. Use delimiters based on the following rules: • Separate time periods with commas `(,)`. • Separate the start time and end time of each time period with a hyphen and a greater-than sign `(->)`.
partitionedJoin	Specifies whether to cache full data of a dimension table in the memory of each concurrent job.	This parameter is optional. Default value: false. This value indicates that full data of the dimension table is cached in the memory of each concurrent job. If you set this parameter to true, partial data of the dimension table is cached in the memory of each concurrent task.

Sample code

The following sample code describes how to create a MaxCompute dimension table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE datahub_input1 (
  id    BIGINT,
  name  VARCHAR,
  age   BIGINT
) with (
  type='datahub'
);


CREATE TABLE odps_dim (
   name VARCHAR,
   phoneNumber BIGINT,
   PRIMARY KEY (name),
   PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
   type = 'odps',
   endPoint = '<yourEndpointName>',
   project = '<yourProjectName>',
   tableName = '<yourTableName>',
   accessId = '<yourAccessId>',
   accessKey = '<yourAccessPassword>',
   `partition` = 'ds=20180905',-- For more information about dynamic or static partitio
ns, see the description for the parameters in the WITH clause.
   cache = 'ALL'
);

CREATE table result_infor(
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
)with(
  type='print'
);

INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN odps_dim FOR SYSTEM_TIME AS OF PROCTIME() as w --You must include this clause when
you perform a JOIN operation on a dimension table.
ON t.name = w.name;
```

Data type mappings

The following table lists the mappings between the data types of MaxCompute and Realtime Compute for Apache Flink (fully-managed Flink). We recommend that you declare the mapping in a DDL statement.

Data type of MaxCompute	Data type of BLINK

TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

> **Note**
> Realtime Compute for Apache Flink supports only the preceding data types in MaxCompute dimension tables.

FAQ

- Q: What is the difference between `max_pt()` and `max_pt_with_done()`?

  `max_pt()` indicates that the partition ranked first in alphabetical order among all partitions is loaded. If the values of the partition parameter are sorted in alphabetical order, `max_pt_with_done()` returns the partition that ranks first in alphabetical order and ends with the `.done` suffix.

Partition

ds=20190101
ds=20190101.done
ds=20190102
ds=20190102.done
ds=20190103

Differences between `max_pt()` and `max_pt_with_done()`:

- `` `partition`='max_pt_with_done()' `` returns the `ds=20190102` partition.
- `` `partition`='max_pt()' `` returns the `ds=20190103` partition.

> **Note**
> Only Blink 3.3.2 and later support `'partition' = 'max_pt_with_done()'`.

- Q: The failover message `RejectedExecutionException: Task java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTas` is reported when a job is running. What do I do?

  A: Dimension table joining in Blink 1.0 has some issues. We recommend that you upgrade the Blink version to 2.1.1 or later. If you still want to use Blink 1.0, you must suspend your job and then resume it. You can troubleshoot this issue based on the first error message in the failover history.

- Q: What do the endPoint and tunnelEndpoint parameters mean in the Alibaba Cloud public cloud? What happens if the two parameters are incorrectly configured?

  A: For more information about the endPoint and tunnelEndpoint parameters, see Endpoints. If the configuration of these two parameters is incorrect in a VPC, one of the following task exceptions may occur.

  - If the endPoint parameter is incorrectly configured, the task stops at a progress of 91%.
  - If the tunnelEndpoint parameter is incorrectly configured, the task fails.

- Q: What do I do if the error message "ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'" appears when a job is running?

  A: This error occurs because the user identity information specified in the MaxCompute DDL statements cannot be used to access MaxCompute. Therefore, you must use an Alibaba Cloud account, a RAM user, or a RAM role to authenticate the user identity. For more information, see User authentication.

5.6.4.7. Create an ApsaraDB for Redis dimension table

This topic describes how to create an ApsaraDB for Redis dimension table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH and CACHE clauses, data type mapping, and sample code used when you create such a dimension table.

> **Important**
> - This topic applies only to Blink 3.2.2 and later.
> - ApsaraDB for Redis dimension tables in Realtime Compute for Apache Flink can only reference data of the STRING type in ApsaraDB for Redis databases.
> - ApsaraDB for Redis dimension tables in Realtime Compute for Apache Flink support user-created Redis databases.

Syntax

In Realtime Compute for Apache Flink, you can create an ApsaraDB for Redis dimension table. The following code shows an example:

```
CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id),  -- The Row Key field in an ApsaraDB for Redis database.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) WITH (
  type = 'redis',
  host = '<yourHostName>',
  port = '<yourPort>',
  password = '<yourPassword>',
  dbNum = '<yourDatabaseNumber>'
);
```

> **Note**
> - Only one primary key can be declared for an ApsaraDB for Redis dimension table.
> - When you join a dimension table with another dimension table, the ON condition must contain equivalent conditions of all the primary keys.
> - You can declare only two fields for an ApsaraDB for Redis dimension table, and the fields must be of the VARCHAR type.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `redis`.
host	The endpoint of an ApsaraDB for Redis instance.	Yes	None.
port	The port of the ApsaraDB for Redis database.	No	Default value: 6379.
dbNum	The sequence number of the ApsaraDB for Redis database.	No	Default value: 0.

password	The password that is used to access the ApsaraDB for Redis database.	No	This parameter is empty by default, which indicates that permission verification is not required.
hashName	The hash key name in hash mode.	No	This parameter is empty by default, which indicates that Realtime Compute for Apache Flink reads data of the STRING type from the ApsaraDB for Redis database. In typical cases, the data type in the Redis dimension table is STRING, which is represented as `key-value` pairs. If you set the **hashName** parameter, data in the ApsaraDB for Redis dimension table is of the HASHMAP type, which is presented as `key-{field-value}` pairs. • **key** is the value of the **hashName** parameter. • **field** is the value of the **key** parameter that you specify in the CREATE TABLE statement. • **value** is the value assigned to **key**, which has the same meaning as **value** in `key-value` of the STRING type.

Parameters in the CACHE clause

Parameter	Description	Required	Remarks
cache	The policy for caching data.	No	Valid values: • None (default value): indicates that no data is cached. • LRU: indicates that partial data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters.
cacheSize	The maximum number of data records that can be cached. Unit: lines.	No	If you set the cache parameter to `LRU`, you can set this parameter to specify the maximum number of data records that can be cached. Default value: 10000.
cacheTTLMs	The cache timeout period.	No	The cache does not expire by default. Unit: milliseconds. If the cache policy is set to LRU, this parameter specifies the time before the cache expires.
cacheEmpty	Specifies whether to clear the cache.	No	Default value: true.

Field type mapping

The following table describes the mapping between ApsaraDB for Redis data types and data types of Realtime Compute for Apache Flink. We recommend that you declare the mapping in a DDL statement.

Data type of ApsaraDB for Redis	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

Sample code

The following sample code demonstrates how to create an ApsaraDB for Redis dimension table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE event (
  id VARCHAR,
  data VARCHAR) with (
  type = 'random'
);

CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id),  -- The Row Key field in an ApsaraDB for Redis database.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) WITH (
  type = 'redis',
  host = '<yourRedisHost>',
  password = '<yourRedisPassword>'
);

SELECT e.*, w. *
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

5.6.4.8. Create an Elasticsearch dimension table

This topic describes how to create an Elasticsearch dimension table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause and CACHE clauses used when you create an Elasticsearch dimension table.

> **Important** This topic applies only to Blink 3.2.2 and later.

DDL syntax

In Realtime Compute for Apache Flink, you can use an Elasticsearch table as a dimension table. The following code shows an example:

```
CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type ='elasticsearch',
  endPoint = '<yourEndPoint>',
  accessId = '<yourUsername>',
  accessKey = '<yourPassword>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

> **Note** An Elasticsearch dimension table supports data updates based on the primary key of an Elasticsearch cluster. You can specify only one field for the primary key field.

Parameters in the WITH clause

Parameter	Description	Default value	Required
type	The type of the dimension table.	elasticsearch	Yes
endPoint	The endpoint of the Elasticsearch cluster, for example, http://127.0.0.1:9211.	No default value	Yes
accessId	The AccessKey ID that is used to access the Elasticsearch cluster.	No default value	Yes
accessKey	The AccessKey secret that is used to access the Elasticsearch cluster.	No default value	Yes
index	The index name, which is similar to the database name.	No default value	Yes
typeName	The type name, which is similar to the database table name.	No default value	Yes
maxRetryTimes	The maximum number of retries for writing data to the table.	30	No
timeout	The read timeout period. Unit: milliseconds.	600000	No
discovery	Specifies whether node discovery is enabled. If this feature is enabled, the client refreshes the server list every 5 minutes.	false	No
compression	Specifies whether to compress request bodies in the GZIP format.	true	No

multiThread	Specifies whether to enable multithreading for JestClient.	true	No

Parameters in the CACHE clause

Parameter	Description	Remarks
cache	The cache policy.	• None: indicates that no data is cached. This value is the default value. • LRU: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record from the source table. If the system does not find the record in the cache, the system searches for the data record in the physical dimension table. If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters. • ALL: indicates that all the data in the dimension table is cached. Before Realtime Compute for Apache Flink runs a job, Realtime Compute for Apache Flink loads all the data in the dimension table to the cache and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the key does not exist. The system reloads all data in the cache after cache entries expire.
cacheSize	The cache size.	You can specify this parameter only if you set the cache parameter to `LRU`. Default value: 10000.
cacheTTLMs	The interval at which the cache is refreshed.	The cache does not time out by default. The purpose of setting this parameter varies based on the cache policy. • If the cache parameter is set to LRU, the cacheTTLMs parameter specifies the timeout period of the cache. • If the cache parameter is set to ALL, the cacheTTLMs parameter specifies the interval at which the cache is refreshed. The cache is not refreshed by default.

5.6.4.9. Create a Phoenix5 dimension table

This topic describes how to create a Phoenix5 dimension table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH and CACHE clauses used when you create a Phoenix5 dimension table.

> **Important** Only Blink versions later than Blink 3.4.0 support Phoenix5 dimension tables.

Syntax

```
create table US_POPULATION_DIM (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<YourServerUrl>',
  tableName = '<YourTableName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `PHOENIX5`.
serverUrl	The URL of the Phoenix5 query server. • If Phoenix5 is created in a cluster, the value of this parameter is the URL of Server Load Balancer (SLB). • If Phoenix5 is created on a single server, the value of this parameter is the URL of the server.	Yes	You must enable the HBase SQL service in an ApsaraDB for HBase instance. The value of the serverUrl parameter is in the **http://host:port** format. In the format: • host: indicates the domain name of the Phoenix5 service. • port: indicates the port number of the Phoenix5 service. Set the value to 8765.
tableName	The name of the Phoenix5 table.	Yes	The name of the Phoenix5 table is in the SchemaName.TableName format. In the format: • SchemaName: indicates the schema name, which can be empty. This means that the schema name is not used and only the table name is used. In this case, the default schema of the database is used. • TableName: indicates the name of the table.

Parameters in the CACHE clause

Parameter	Description	Required	Remarks

cache	The cache policy.	No	Valid values: • None: indicates that no data is cached. This is the default value. • LRU: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • ALL: indicates that all the data in the dimension table is cached. Before Realtime Compute for Apache Flink runs a job, Realtime Compute for Apache Flink loads all the data in the dimension table to the cache and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the data amount of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters. **Note** If you set the cache parameter to ALL, you must increase the memory of the node for joining tables because Realtime Compute for Apache Flink asynchronously loads data from the dimension table. The increased memory size is twice that of the remote table.
cacheSize	The maximum number of data records that can be cached.	No	This parameter is available only if you set the cache parameter to LRU. Default value: 10000. Unit: rows.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	If the cache parameter is set toLRU, the cacheTTLMs parameter specifies the time allowed before cache entries expire. Cache entries do not expire by default. If the cache parameter is set to ALL, the cacheTTLMs parameter specifies the interval at which the cache is loaded. The cache is not reloaded by default.

cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter takes effect when the cache parameter is set to ALL. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	No	This parameter is optional. It is empty by default. For example, you can set this parameter to **'2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'**. Multiple time periods are separated by commas (,). The start time and end time of each time period are separated with a hyphen and a greater-than sign (->).

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--Define the change period of the dimension table.
)with(
  type='PHOENIX5'
);

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 operation on the dimension table.
ON t.name = w.name;
```

5.6.4.10. Create an AnalyticDB for MySQL V3.0 dimension table

This topic describes how to create an AnalyticDB for MySQL V3.0 dimension table. This topic also describes the parameters in the WITH clause and cache parameters used when you create an AnalyticDB for MySQL V3.0 dimension table.

> **Important** This topic applies only to **Blink-3.5.0-hotfix** and later.

DDL syntax

```
CREATE TABLE dim_ads(
    `name` VARCHAR,
    id VARCHAR,
    PRIMARY KEY (`name`),
    PERIOD FOR SYSTEM_TIME
)with(
    type='ADB30',
    url='jdbc:mysql://<Internal endpoint>/<databaseName>',
    tableName='xxx',
    userName='xxx',
    password='xxx'
);
```

> **Note**
> - You must specify a primary key when you declare a dimension table.
> - When you join a dimension table with another table, the ON condition must contain equality conditions that include all primary keys.
> - The primary key of an AnalyticDB for MySQL database can be defined as the primary key or unique index column of an AnalyticDB for MySQL dimension table.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to ADB30.

url	The URL of the AnalyticDB for MySQL database.	Yes	The URL of the AnalyticDB for MySQL database, such as `url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'`. **Note** • For more information about how to access an AnalyticDB for MySQL database, see Register an AnalyticDB for MySQL instance. • databaseName indicates the name of the AnalyticDB for MySQL database.
tableName	The name of the table.	Yes	N/A.
userName	The username that is used to access the AnalyticDB for MySQL database.	Yes	N/A.
password	The password that is used to access the AnalyticDB for MySQL database.	Yes	N/A.
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 3.

Cache parameters

Parameter	Description	Required	Remarks
			Valid values: • None: indicates that data is not cached. This is the default cache policy. • LRU: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record from the source table. If the system does not find the record in the cache, the system searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • ALL: indicates that all the data in the dimension table is cached. Before the system runs a job, the system loads all data in the dimension table to

cache	The policy that is used to cache data.	No	the cache. This way, the cache is searched for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data in a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. The source table and dimension table cannot be associated based on the ON clause. If you use this cache policy, you must configure the cacheTTLMs and cacheReloadTimeBlackList parameters. **Note** • If you set the cache parameter to ALL, you must increase the memory of the node for joining tables because the system asynchronously loads data from the dimension table. The increased memory size is twice the memory size of the remote table. • If a dimension table stores a large volume of data and the cache parameter is set to ALL, an out of memory (OOM) error may occur or a full garbage collection (GC) may be time-consuming. To address this issue, you can use one of the following methods: ◦ If the cache parameter can be set to ALL for a dimension table, enable the partitionedJoin feature. For a Blink version earlier than Blink 3.6.0, the full data of the dimension table is loaded for each concurrent job by default. For a Blink version later than Blink 3.6.0, the partitionedJoin feature is available if you set the cache parameter to ALL. After you enable the partitionedJoin feature, only the required data is cached for each concurrent job. ◦ Use an ApsaraDB for HBase or ApsaraDB RDS dimension table that uses key-value pairs to store data.
cacheSize	The maximum number of rows of data records that can be cached.	No	This parameter is available only if you set the cache parameter to LRU. Default value: 10000.

cacheTTLMs	The interval at which the system refreshes the cache. The system reloads the latest data in the dimension table based on the value of this parameter. This ensures that the data in the source table is associated with the latest data in the dimension table.	No	Unit: milliseconds. This parameter is empty by default. This indicates that the updates in the dimension table are not reloaded.
cacheReloadTimeBlackList	The periods of time during which cache is not refreshed. This parameter takes effect when the cache parameter is set to ALL. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	No	This parameter is optional. This parameter is empty by default. For example, you can specify this parameter as '**2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00**'. Use the following delimiters to separate time periods: • Separate multiple time periods with commas (,). • Separate the start time and end time of each time period with a hyphen and a greater-than sign (->).
partitionedJoin	Specifies whether to enable the partitionedJoin feature. If the partitionedJoin feature is enabled, shuffling is implemented based on join keys before the primary table is joined with the dimension table. This process provides the following benefits: • If you set the cache parameter to LRU, the cache hit rate increases. • If you set the cache parameter to ALL, memory resources are reduced because only the required data is cached for each concurrent job.	No	The default value of this parameter is false. This indicates that the partitionedJoin feature is disabled. **Note** Before you enable the partitionedJoin feature, set **partitionedJoin** to true.

maxJoinRows	The maximum number of results that are returned each time a data record in the primary table is queried and matched with data records in the dimension table.	No	Default value: 1024. If you can estimate that a data record in the primary table corresponds to a maximum of n data records in the dimension table, you can set the **maxJoinRows** to n to ensure efficient matching in Realtime Compute for Apache Flink. **Note** When you join a dimension table with another table, this parameter specifies the maximum number of results that can be returned after a data record in the primary table is matched with data records in the dimension table.

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--The identifier of a dimension table.
) with (
  type='ADB30'
);

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
when you perform a JOIN operation on the dimension table.
ON t.name = w.name;
```

5.6.4.11. Create an Oracle database dimension table

This topic describes how to create an Oracle database dimension table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, field type mappings, and attribute fields used when you create an Oracle database dimension table.

DDL syntax

```
CREATE TABLE oracle_dim(
  employee_id BIGINT,
  phone_number BIGINT,
  dollar DOUBLE,
  PRIMARY KEY (employee_id)
) WITH (
  type = 'oracle_dim',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>',
  cache = 'ALL'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Example
type	The type of the dimension table.	Yes	Set the value to oracle_dim.
url	The Java Database Connectivity (JDBC) URL of the Oracle database.	Yes	`jdbc:oracle:thin:@ip:port:sid`
userName	The username that is used to access the database.	Yes	None.
password	The password that is used to access the database.	Yes	None.
tableName	The name of the table.	Yes	None.
maxRetryTimes	The maximum number of retries for reading data from the dimension table.	No	Default value: 10.

Parameters in the CACHE clause

Parameter	Description	Required	Example
cache	The cache policy.	No	Valid values: • None: indicates that no data is cached. This is the default value. • LRU: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters. • ALL: indicates that all the data in the dimension table is cached. Before Realtime Compute for Apache Flink runs a job, Realtime Compute for Apache Flink loads all the data in the dimension table to the cache and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters. **Note** If the cache parameter is set to ALL, the memory of the node for joining tables must be increased because Realtime Compute for Apache Flink asynchronously loads data from the dimension table. The increased memory size is two times that of the remote table.
cacheSize	The cache size, which specifies the number of rows of data that can be cached.	No	You can set this parameter only after you set the cache parameter to LRU. Default value: 10000.

cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	• You can set this parameter only if you set the **cache** parameter to LRU. By default, the cached data does not expire. • If the **cache** parameter is set to ALL, the cacheTTLMs parameter specifies the interval at which Realtime Compute for Apache Flink refreshes the cache. The cache is not refreshed by default.
cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter takes effect when the cache parameter is set to ALL. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	No	This parameter is empty by default. Format of the parameter value: **'2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'**. Take note of the following points when you use delimiters: • Separate time periods with commas (,). • Separate the start time and end time of each time period with a hyphen and a greater-than sign (->).
maxJoinRows	The maximum number of records in the right table that are connected to a record in the left table in a one-to-many join.	No	Default value: 1024. **Note** If a large number of records are connected to a record in a one-to-many join, the cache memory needs to be adjusted. The cacheSize parameter limits the number of keys in the left table. If a single record in the left table corresponds to a large number of records in the right table, the performance of streaming tasks may be significantly affected.

Field type mapping

Data type of Oracle database	Data type of Realtime Compute for Apache Flink
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

Sample code

```
CREATE TABLE oracle_source (
  employee_id BIGINT,
  employee_name VARCHAR,
  employee_age INT
) WITH (
  type ='random'
);


CREATE TABLE oracle_dim (
  employee_id BIGINT,
  phone_number BIGINT,
  dollar DOUBLE,
  PRIMARY KEY (employee_id)
) WITH (
  type = 'oracle_dim',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>',
  cache = 'ALL'
);

CREATE TEMPORARY TABLE oracle_sink (
  employee_id BIGINT,
  phone_number BIGINT,
  employee_name VARCHAR
) WITH (
  type = 'oracle',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>'
);

INSERT INTO oracle_sink
SELECT t.employee_id, w.phone_number, t.employee_name
FROM oracle_source as t JOIN oracle_dim FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.employee_id = w.employee_id;
```

5.7. DML statement

5.7.1. EMIT statements

You can execute EMIT statements to define different output policies for a query in different scenarios. This allows you to control delays and improves data accuracy.

> **Important** Only Realtime Compute for Apache Flink V2.0.0 and later support EMIT statements.

Limits

- Only tumbling and sliding windows support EMIT policies. Session windows do not support EMIT policies.
- If a job has multiple outputs, you must define the same EMIT policy for the outputs. You will be able to define different EMIT policies for the outputs in the future.
- The EMIT syntax cannot be used to set the allowLateness parameter for minibatch. You will be able to declare allowLateness in EMIT policies in the future.

EMIT policies

An EMIT policy is an output policy for a query in a specific scenario of Flink SQL. For example, an output policy may specify the maximum delay for outputs. The traditional ANSI SQL syntax does not support the output policies of this type. Assume that you want to view the latest result every minute before a 1-hour window ends and do not want to lose the data that arrives within one day after the window ends. If the collected statistics do not change before the window ends, the output result is not updated. If the collected statics change before the window ends, the output result is updated.

Based on this scenario, Realtime Compute for Apache Flink introduces the EMIT syntax and extends SQL statements with the EMIT syntax. In the following example, EMIT policies are defined for different scenarios:

- Before a window ends, results are generated with a 1-minute delay. After the window ends, results are generated without delays.

```
EMIT
  WITH DELAY '1'MINUTE BEFORE WATERMARK,
  WITHOUT DELAY AFTER WATERMARK
```

- Before a window ends, no result is generated. After the window ends, results are generated without delays.

```
EMIT WITHOUT DELAY AFTER WATERMARK
```

- Results are generated with a 1-minute delay. You can set the minibatch parameter to increase the delay.

```
EMIT WITH DELAY '1'MINUTE
```

- Before the window ends, results are generated with a 1-minute delay.

```
EMIT WITH DELAY '1'MINUTE BEFORE WATERMARK
```

Usage

The EMIT syntax offer the following features:

- Controls delays. You can set the output frequency before the window ends to decrease the delay of displaying results.
- Improves data accuracy. The system does not discard the data that arrives late. This ensures the accuracy of outputs.

> **Note** When you specify an EMIT policy, you must balance between business complexity and resource consumption. A lower output delay and a higher data accuracy require a higher computing overhead.

Syntax

You can use the EMIT syntax in the INSERT INTO statement to define a policy for outputs. If you do not use the EMIT syntax in the INSERT INTO statement, the default setting takes effect. By default, a window generates a result only when the window ends. A watermark is

triggered when the window ends.

> **Note** You can append an EMIT statement only to the end of the query statements in the INSERT INTO statement. You cannot include an EMIT statement in a VIEW statement.

```
INSERT INTO tableName
<Query>
EMIT strategy [, strategy]*

strategy ::= {WITH DELAY timeInterval | WITHOUT DELAY}
               [BEFORE WATERMARK |AFTER WATERMARK]

timeInterval ::='string' timeUnit
```

Parameter	Description
`WITH DELAY`	Specifies the maximum output delay. Results are generated at the specified interval.
`WITHOUT DELAY`	Specifies that no delay is allowed. A result is immediately generated after each data record arrives.
`BEFORE WATERMARK`	Specifies the policy that is used before a window ends. A watermark is triggered when a window ends.
`AFTER WATERMARK`	Specifies the policy that is used after a window ends. A watermark is triggered when a window ends. **Note** If you configure the AFTER WATERMARK policy, you must explicitly declare the `blink.state.ttl.ms` parameter to specify the maximum delay.

You can use the following `strategy` settings:

- Set it to one BEFORE policy.
- Set it to one AFTER policy.
- Set it to one BEFORE policy and one AFTER policy.

> **Note** You cannot define two BEFORE policies or two AFTER policies at the same time for `strategy`.

TTL

If the AFTER WATERMARK policy is configured, the information about the window state is retained for a specified period to wait for late data. The retention period is called time to live (TTL). After the AFTER policy is applied, you can explicitly declare the `blink.state.ttl.ms` parameter to set the TTL for the information about the window state. For example, `blink.state.ttl.ms = 3600000` means that the window can wait for late data for up to 1 hour. The data that arrives more than 1 hour late is discarded.

Examples

A 1-hour tumbling window is used as an example. The following code block describes the `tumble_window` syntax:

```
CREATE VIEW tumble_window AS
SELECT
  `id`,
  TUMBLE_START(rowtime, INTERVAL '1' HOUR) as start_time,
  COUNT(*) as cnt
FROM source
GROUP BY `id`, TUMBLE(rowtime, INTERVAL '1' HOUR);
```

By default, you must wait for 1 hour before you can obtain the `tumble_window` result. If you need to view the latest result of the window every minute even if the result is incomplete, you can execute the following statements:

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK; --Before the window ends, the updated resu
lt is generated at 1-minute intervals.
```

By default, `tumble_window` ignores and discards the data that arrives after the window ends. In some scenarios, you may want the outputs to include the data that arrives one day after the window ends. You may also want the results to be immediately updated after each data record is received. To meet these requirements, you can execute the following statements:

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK,
     WITHOUT DELAY AFTER WATERMARK; --After the window ends, updated results are
immediately generated after a data record is received.
```

In addition, you must set **blink.state.ttl.ms** to 86400000 in the **job parameters**. This value indicates that the window can wait for late data for up to one day.

Delay

In an EMIT policy, `DELAY` specifies the maximum allowed duration. The duration starts from the time when your data flows into Realtime Compute for Apache Flink and ends at the time when you obtain the result data. The end time is an event time or a processing time. A delay is calculated based on the system time. The delay is the interval between the time when data changes in a dynamic table and the time when a new data record is displayed in a result table. The dynamic table stores streaming data in Realtime Compute for Apache Flink. The result table is referenced by Realtime Compute for Apache Flink and is stored in an external data store.

If the processing time in Realtime Compute for Apache Flink is 0, a delay may occur when streaming data accumulates and when a window waits for data. If you specify a maximum delay of 30 seconds, streaming data can be accumulated during the 30 seconds. If a 1-hour window is specified in a query, a maximum delay of 30 seconds indicates that the output results are updated every 30 seconds.

- Use the configuration `EMIT WITH DELAY '1' MINUTE` as an example.

  When you use the GROUP BY clause to aggregate data, the system accumulates streaming data within 1 minute. For a window whose size is greater than 1 minute, the window generates a result every 1 minute. If the size of the window is less than 1 minute, the system ignores this configuration. This is because watermarks can be used to meet the delay requirements in the service level agreement (SLA) for window outputs.
- Use the configuration `EMIT WITHOUT DELAY` as an example.

When you use the GROUP BY clause to aggregate data, the system does not use the minibatch parameter to increase the delay. Each data record is immediately processed after it is received. The processing result is also immediately generated. When you use window functions, each data record is immediately processed after it is received. The processing result is also immediately generated.

5.7.2. INSERT INTO statements

This topic describes the method and limits of executing INSERT INTO statements in Realtime Compute for Apache Flink.

Operation limits

Table type	Limit
Source table	Can be referenced in only FROM clauses and does not support INSERT statements.
Dimension table	Can be referenced in only JOIN statements and does not support INSERT statements.
Result table	Supports only INSERT statements.
View	Can be referenced in only FROM clauses.

Syntax

```
INSERT INTO tableName
[ (columnName[ , columnName]*) ]
queryStatement;
```

Examples

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;

INSERT INTO Orders(z,v)
SELECT c,d FROM OO;
```

Note

- In Realtime Compute for Apache Flink, a single SQL job can contain multiple data manipulation language (DML) operations, data sources, data destinations, and dimension tables. For example, a job file can contain two snippets of SQL statements for independent services. You can execute the SQL statements to write data to different data destinations.
- Realtime Compute for Apache Flink does not allow you to execute a separate SELECT statement to query data. To execute a SELECT statement, you must include the SELECT statement in a CREATE VIEW or an INSERT INTO statement.
- You can execute an INSERT INTO statement to update an existing record. For example, you can insert a key value into an ApsaraDB for RDS result table that contains a primary key. If the key value already exists, the existing record is updated. If the key value does not exist, a new key value is inserted.

5.8. Query statements

5.8.1. SELECT statements

You can execute SELECT statements to retrieve data from tables.

Syntax

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

Test data

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

Simple queries

- Test statement

```
SELECT * FROM <Table name>;
```

- Test result

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

Rename objects

- Test statement

```
SELECT a, c AS d FROM <Table name>;
```

- Test result

a (VARCHAR)	d (DATE)
a1	1990-02-20
b1	2018-05-12

c1	2010-06-14
a1	2016-04-05

Deduplication queries

- Test statement

```
SELECT DISTINCT a FROM Table name;
```

- Test result

a (VARCHAR)
a1
b1
c1

Subqueries

In most cases, SELECT statements read data from tables, for example, `SELECT column_1, column_2 … FROM table_name`. SELECT statements can also read data from the results of other SELECT statements. This is known as subqueries.

> **Note** You must specify aliases in subqueries.

- Test statement

```
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;
```

- Test result

a (VARCHAR)	b (INT)
a1	211
b1	120
a1	257

> **Note** The preceding test result is a debugging result. In the result, you can view the computing process. If your job is published and the result table is stored in DataHub, Alibaba Cloud Message Queue for Apache Kafka, or Alibaba Cloud Message Queue, the computing process is displayed. If your job is published and the result table is stored in a relational database such as ApsaraDB RDS, the records that have the same primary key values are combined into one record.

5.8.2. WHERE

A WHERE clause filters data returned by a SELECT statement.

Syntax

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ];
```

The following table describes the operators that can be used in a WHERE clause.

Operator	Description
=	Equal to
<>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Example

- Test data

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	Shanghai

- Test statements

```
SELECT * FROM XXXX WHERE City='Beijing';
```

- Test results

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

5.8.3. HAVING statement

When using an aggregate function, you need to add a HAVING statement to achieve the same filtering effect as a WHERE statement.

Syntax

```
SELECT [ ALL | DISTINCT ]{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ];
```

Example

- Test data

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- Test statement

```
SELECT Customer,SUM(OrderPrice) FROM XXX
GROUP BY Customer
HAVING SUM(OrderPrice)<2000;
```

- Test result

Customer	SUM(OrderPrice)
Carter	1700

5.8.4. GROUP BY statement

A GROUP BY statement groups a result set by one or more columns.

Syntax

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ GROUP BY { groupItem [, groupItem ]* } ];
```

Example

- Test data

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- Test statement

```
SELECT Customer,SUM(OrderPrice) FROM xxx
GROUP BY Customer;
```

- Test result

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

5.8.5. JOIN statements

JOIN statements in Realtime Compute for Apache Flink have the same semantic meanings as those for batch processing. The two types of JOIN statements allow you to join two tables. The difference is that each JOIN statement in Realtime Compute for Apache Flink joins two dynamic tables. The join results are dynamically updated to ensure that the final results are the same as the corresponding results of batch processing.

Syntax

```
tableReference [, tableReference ]* | tableexpression
[ LEFT ] JOIN tableexpression [ joinCondition ];
```

- tableReference: specifies the table name.
- tableexpression: specifies the expression.
- joinCondition: specifies the join condition.

> **Important**
> - Only EQUI JOIN operations are supported.
> - Only INNER JOIN and LEFT OUTER JOIN operations are supported.

Example 1: Join the Orders table and the Products table

- Test data Table 1. Orders

rowtime	productId	orderId	units
`10:17:00`	30	5	4
`10:17:05`	10	6	1
`10:18:05`	20	7	2
`10:18:07`	30	8	20
`11:02:00`	10	9	6
`11:04:00`	10	10	1
`11:09:30`	40	11	12
`11:24:11`	10	12	4

Table 2. Products

productId	name	unitPrice
30	Cheese	`17`
10	Beer	`0.25`
20	Wine	`6`
30	Cheese	`17`
10	Beer	`0.25`
10	Beer	`0.25`
40	Bread	`100`

10	Beer	0.25

- Test statement

```
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;
```

- Test result

o.rowtime	o.productId	o.orderId	o.units	p.name	p.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: Join the datahub_stream1 table and the datahub_stream2 table

- Test data Table 3. datahub_stream1

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21

Table 4. datahub_stream2

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21
0	10	test31
1	10	test41

- Test statement

```
SELECT s1.c,s2.c
FROM datahub_stream1 AS s1
JOIN datahub_stream2 AS s2
ON s1.a =s2.a
WHERE s1.a = 0;
```

- Test result

s1.c (VARCHAR)	s2.c (VARCHAR)
test11	test11
test11	test31

5.8.6. JOIN statements for dimension tables

In Realtime Compute for Apache Flink, each data stream can be associated with a dimension table that is stored in an external data source. This allows you to perform associated queries in Realtime Compute for Apache Flink.

> **Note** A dimension table constantly changes. Therefore, when you associate a data stream with a dimension table, you must specify the time of the dimension table snapshot with which the data stream is associated. A data stream can be associated only with the dimension table snapshot that is taken at the current time. In the future, Realtime Compute for Apache Flink will allow you to associate data streams with dimension table snapshots that are taken at different points in time. The points in time are specified by the rowtime field in the left table. For more information about dimension tables, see Overview.

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;
```

The following example shows an event stream that is joined with a whitelist dimension table.

```
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

Note

- Dimension tables support `INNER JOIN` and `LEFT JOIN` operations, and do not support `RIGHT JOIN` or `FULL JOIN` operations.
- You must append `FOR SYSTEM_TIME AS OF PROCTIME()` to the end of the dimension table. This way, each data record in the dimension table that can be viewed at the current time is associated with the source data.
- The subsequent input data in the source table is associated only with the latest records that are stored in the dimension table at the current time. This means that the JOIN operation is performed only at the processing time. Therefore, if the data in the dimension table is added, updated, or deleted after the JOIN operation is performed, the associated data remains unchanged.
- The ON clause must contain equivalent (=) conditions for all primary key fields of the dimension table. The primary key fields in the conditions must be the same as those in the physical tables that are referenced in the SQL statement. The ON clause can also contain other equivalent (=) conditions.
- If you want to perform one-to-many table joins, you must specify the join keys in the data definition language (DDL) INDEX syntax for dimension tables. For more information, see INDEX syntax.
- Two dimension tables cannot be joined.
- In the join conditions that are specified in the ON clause, the fields in the dimension table cannot use type conversion functions, such as CAST. If you need to convert data types, perform the conversion on the fields in the source table.

Example

- Test data Table 1. datahub_input1

id (BIGINT)	name (VARCHAR)	age (BIGINT)
1	lilei	22
2	hanmeimei	20
3	libai	28

Table 2. phoneNumber

name (VARCHAR)	phoneNumber (BIGINT)
dufu	13900001111
baijuyi	13900002222
libai	13900003333
lilei	13900004444

- Test statements

```
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;
```

- Test results

id (BIGINT)	phoneNumber (BIGINT)	name (VARCHAR)
1	13900004444	lilei
3	13900003333	libai

5.8.7. IntervalJoin statement

The IntervalJoin statement allows two streams to be joined. During the JOIN operation, each record in the left and right streams is associated with only data generated at the same time in the other stream. After the streams are joined, the time column in the input stream is still retained for you to continue to perform operations based on the event time.

Syntax

```
SELECT column-names
FROM table1  [AS <alias1>]
[INNER | LEFT | RIGHT |FULL ] JOIN table2
ON table1.column-name1 = table2.key-name1 AND TIMEBOUND_EXPRESSION
```

Note

- INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN are supported. If you use JOIN directly, INNER JOIN is automatically used.
- SEMI JOIN or ANTI JOIN is not supported.
- TIMEBOUND_EXPRESSION is an interval conditional expression on the columns of the time attributes of the left and right streams. The following conditional expressions are supported:
  - **ltime = rtime**
  - **ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE**
  - **ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND**

Example 1 (Event time-based)

This example shows the statistics on the logistics information within 4 hours after orders are placed.

- Test data
  - Order table (orders)

id	productName	orderTime
1	iphone	2020-04-01 10:00:00.0
2	mac	2020-04-01 10:02:00.0
3	huawei	2020-04-01 10:03:00.0
4	pad	2020-04-01 10:05:00.0

  - Logistics table (shipments)

shipId	orderId	status	shiptime
0	1	shipped	2020-04-01 11:00:00.0
1	2	delivered	2020-04-01 17:00:00.0
2	3	shipped	2020-04-01 12:00:00.0
3	4	shipped	2020-04-01 11:30:00.0

- Test statements

```
CREATE TABLE Orders(
  id BIGINT,
  productName VARCHAR,
  orderTime TIMESTAMP,
  WATERMARK wk FOR orderTime as withOffset(orderTime, 2000)  --Define a watermark for
the rowtime.
) WITH (
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);

CREATE TABLE Shipments(
  shipId BIGINT,
  orderId BIGINT,
  status VARCHAR,
  shiptime TIMESTAMP,
  WATERMARK wk FOR ts as withOffset(shiptime, 2000) --Define a watermark for the rowt
ime.
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);

--Create an ApsaraDB RDS result table.
CREATE TABLE rds_output(
  id BIGINT,
  productName VARCHAR,
  status VARCHAR
) WITH (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='<yourDatabaseTablename>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);

INSERT INTO rds_output
SELECT id, productName, status
FROM Orders AS o
JOIN Shipments AS s on o.id = s.orderId AND
     o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime;
```

- Test result

id(bigint)	productName(varchar)	status(varchar)
1	iphone	shipped
3	huawei	shipped
4	pad	shipped

Example 2 (Processing time-based)

- Test data
  - datahub_stream1

k1	v1
1	val1
2	val2
3	val3

  - datahub_stream2

k1	v1
1	val1
2	val2
3	val3

- Test statements

```
CREATE TABLE datahub_stream1 (
  k1 BIGINT,
  v1 VARCHAR,
  d AS PROCTIME()
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);

CREATE TABLE datahub_stream2 (
  k2 BIGINT,
  v2 VARCHAR,
  e AS PROCTIME()
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);

--Create an ApsaraDB RDS result table.
CREATE TABLE rds_output(
  k1 BIGINT,
  v1 VARCHAR,
  v2 VARCHAR
) WITH (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='<yourDatabaseTablename>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);

INSERT INTO rds_output
SELECT k1, v1, v2
FROM datahub_stream1 AS o
JOIN datahub_stream2 AS s on o.k1 = s.k2 AND
     o.d BETWEEN s.e - INTERVAL '4' MINUTE AND s.e;
```

Note The results are uncertain because they depend on the time when each data record in the two data streams enters the system. Therefore, this example does not provide the expected results.

5.8.8. UNION ALL

A UNION ALL clause is used to combine two data streams. The field types and sequences of the two data streams must be the same.

Syntax

```
select_statement
UNION ALL
select_statement;
```

> **Note** Realtime Compute for Apache Flink also supports the `UNION` function. `UNION ALL` allows duplicate values and `UNION` does not allow duplicate values. In Realtime Compute for Apache Flink, `UNION` is equivalent to the combination of `UNION ALL and DISTINCT`. We recommend that you do not use `UNION` because its operating efficiency is low.

Example

- Test data Table 1. test_source_union1

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10

Table 2. test_source_union2

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10
test2	2	20

Table 3. test_source_union3

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10
test2	2	20
test1	1	10

- Sample code

```
SELECT
    a,
    sum(b) as d,
    sum(c) as e
FROM
    (SELECT * from test_source_union1
    UNION ALL
    SELECT * from test_source_union2
    UNION ALL
    SELECT * from test_source_union3
    )t
 GROUP BY a;
```

- Test results

a (VARCHAR)	d (BIGINT)	e (BIGINT)
test1	1	10
test2	2	20
test1	2	20
test1	3	30
test2	4	40
test1	4	40

> **Note** The preceding test results are debugging results. In these results, you can view the computing process. If your job is published and the result table is stored in DataHub, Alibaba Cloud Message Queue for Apache Kafka, or Alibaba Cloud Message Queue, the result data contains data about the computing process. If your job is published and the result table is stored in a relational database such as ApsaraDB RDS, the records that have the same primary key values are combined into one record.

5.8.9. TopN

A TopN clause is used to compute the largest or smallest N data records of a metric in real-time data. Flink SQL uses an OVER window function to flexibly implement TopN computing.

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]
```

> **Note**
> - `ROW_NUMBER()` : specifies an `OVER` window function to compute the row number. The value starts from 1.
> - `PARTITION BY col1[, col2..]` : specifies one or more partitioning columns. This parameter is optional.
> - `ORDER BY col1 [asc|desc][, col2 [asc|desc]...]` : specifies the columns based on which you want to sort data and the sorting order of each column.

As shown in the preceding syntax, a TopN clause requires two levels of queries.

- In the subquery, the `ROW_NUMBER()` window function is used to sort data based on the specified columns and mark the data with rankings.
- In the outer query, only the first N data records in the ranking list are obtained. For example, if N is set to 10, the first 10 data records are obtained.

During the execution, Flink SQL sorts an input data stream based on the sort key. If the first N data records in a partition are changed, the updated data is sent downstream as an update stream.

> **Note** Therefore, if you want to export the TopN data to external storage, the target result table must contain a primary key.

WHERE

To enable Flink SQL to identify a TopN query, use `rownum <= N` in the outer query to specify the first N records. Do not place `rownum` in an expression, for example, `rownum - 5 <= N` . If you specify multiple conditions in the WHERE clause, you must use `AND` to join the conditions.

Example 1

You can use the following example to return the top 100 keywords that are queried the most in each city within a specific hour. The hour, city, and ranking columns in the output table identify a unique record. Therefore, you must declare the three columns as a composite key. The key must also be configured in the external storage.

```
CREATE TABLE rds_output (
  rownum BIGINT,
  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 SUBSTRING(time_str,1,12) AS start_time,
            keyword,
            count(1) AS pv,
            city
        FROM tmp_search
        GROUP BY SUBSTRING(time_str,1,12), keyword, city
    ) a
) t
WHERE rownum <= 100
```

Example 2

- Test data

ip (VARCHAR)	time (VARCHAR)
192.168.1.1	100,000,000
192.168.1.2	100,000,000
192.168.1.2	100,000,000
192.168.1.3	100,030,000
192.168.1.3	100,000,000
192.168.1.3	100,000,000

- Test statements

```
CREATE TABLE source_table (
  IP VARCHAR,
  `TIME` VARCHAR
)WITH(
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>'
);

CREATE TABLE result_table (
  rownum BIGINT,
  start_time VARCHAR,
  IP VARCHAR,
  cc BIGINT,
  PRIMARY KEY (start_time, IP)
) WITH (
  type = 'rds',
  url='<yourDatabaseAddress>',
  tableName='blink_rds_test',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
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,-- You can specify a value based o
n the actual time. The data specified here is an example.
        COUNT(IP) AS cc,
        IP
        FROM  source_table
        GROUP BY SUBSTRING(`TIME`,1,2), IP
    )a
) t
WHERE rownum <= 3 -- You can specify a value based on the number of data records you
want to obtain. The data specified here is an example.
```

- Test results

rownum (BIGINT)	start_time (VARCHAR)	ip (VARCHAR)	cc (BIGINT)
1	10	`192.168.1.3`	3
2	10	`192.168.1.2`	2
3	10	`192.168.1.1`	1

No ranking number optimization

- You can use no ranking number optimization to solve the data bloat issue.
  - Data bloat

    Based on the TopN syntax, the `rownum` field is written into a result table as one of its primary keys. This may lead to data bloat. For example, if the ranking of a data record rises from the ninth to the first after a data update, the rankings of the records from the first to the ninth all change. The changes must be updated in the result table. This results in data bloat. The data update in the result table may slow down because an excessive volume of data is received.
  - Method of no ranking number optimization

    To avoid data bloat, remove `rownum` from the result table and compute `rownum` at the front end. The volume of TopN data records is not large, so the top 100 data records can be obtained quickly at the front end. In this case, if the ranking of a data record rises from the ninth to the first after an update, only this record needs to be delivered to the result table. This accelerates data update in 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]
```

  The syntax is similar to the original TopN syntax. You only need to remove the rownum field from the outer query.

> **Note** If the rownum field is removed, pay attention to the definition of the primary keys in the result table. If the definition is incorrect, the TopN query result is incorrect. The primary keys must be defined in the key list at the GROUP BY node before the TopN clause.

- Example

  This example is a case from a customer in the video industry. Heavy traffic is generated each time a video is distributed. You can identify the most popular videos based on the video traffic. The following example identifies the top 5 videos that generate the most traffic per minute.
  - Test statements

```
-- Read the original data storage table from Log Service.
CREATE TABLE sls_cdnlog_stream (
vid VARCHAR, -- The video ID.
rowtime TIMESTAMP, -- The time when the video was watched.
response_size BIGINT, -- The traffic generated for watching the video.
WATERMARK FOR rowtime as withOffset(rowtime, 0)
) WITH (
type='sls',
...
);

-- Compute the consumed bandwidth by video ID in a 1-minute window.
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);

-- Create a result table.
CREATE TABLE hbase_out_cdnvidtoplog (
vid VARCHAR,
rss BIGINT,
start_time VARCHAR,
   -- Do not store the rownum field in the result table.
   -- Pay attention to the definition of the primary keys. The primary keys must be
defined in the key list at the GROUP BY node before the TopN clause.
PRIMARY KEY(start_time, vid)
) WITH (
type='RDS',
...
);

-- Identify and export the IDs of the top 5 videos that generate the most traffic
per minute.
INSERT INTO hbase_out_cdnvidtoplog

-- Do not include the rownum field in the outer 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;
```

- Test data

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

- Test results

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

5.8.10. GROUPING SETS clause

This topic describes how to use the GROUPING SETS clause in a single SELECT statement to aggregate and analyze data from multiple dimensions. For example, you can use the GROUPING SETS clause to aggregate the data in Column a, Column b, and both columns. If you do not use the GROUPING SETS clause, you must execute multiple UNION ALL clauses for multi-dimensional data aggregation and analysis. In this case, the system performance is compromised.

Syntax

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
GROUP BY
[GROUPING SETS { groupItem [, groupItem ]* } ];
```

Examples

- Test data

username	month	day
Lily	10	1
Lucy	11	21
Lily	11	21

- Sample code

```
SELECT
    `month`,
    `day`,
    count(distinct `username`) as uv
FROM tmall_item
group by
grouping sets((`month`),(`month`,`day`));
```

- Test results

month	day	uv
10	1	1
10	null	1
11	21	1
11	null	1
11	21	2
11	null	2

> **Note** The preceding test result is a debugging result. In the result, you can view the computing process. If your job is published and the result table is stored in DataHub, Alibaba Cloud Message Queue for Apache Kafka, or Alibaba Cloud Message Queue, the result data contains the data about the computing process. If your job is published and the result table is stored in a relational database such as ApsaraDB for RDS, the records that have the same primary key values are combined into one record.

5.8.11. CEP statements

MATCH_RECOGNIZE is a complex event processing (CEP) statement that identifies events from input data streams based on the specified rules and generates output events based on the 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 MATCH 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 clause to divide the rows of an input table into partitions based on one or more partition key columns.
ORDER BY	Optional. You can use the clause to sort the rows of a partition based on one or more columns. If you use multiple columns to sort the rows, specify `EVENT TIME` or `PROCESS TIME` as the first column.
MEASURES	You can use the clause to define the output events that are generated based on the matched input events.
ONE ROW PER MATCH	If you use the clause, only one output event is generated for each match.
ONE ROW PER MATCH WITH TIMEOUT ROWS	If you use the clause, an output event is generated for each match or time-out. The time-out period is specified by the **WITHIN** clause in **PATTERN**. **Note** Blink V3.6.0 and later do not support the **ONE ROW PER MATCH WITH TIMEOUT ROWS** clause due to Calcite upgrades.
ALL ROWS PER MATCH	If you use the clause, an output event is generated for each input event upon each match.
ALL ROWS PER MATCH WITH TIMEOUT ROWS	If you use the clause, an output event is generated for each input event upon each match or time-out. The time-out period is specified by the **WITHIN** clause in **PATTERN**. **Note** Blink V3.6.0 and later do not support the **ALL ROWS PER MATCH WITH TIMEOUT ROWS** clause due to Calcite upgrades.

[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]	Optional. By default, `ONE ROW PER MATCH` is used.
AFTER MATCH SKIP TO NEXT ROW	If you use the clause, the next match starts to be performed from the event that follows the first event in the sequence of matched events.
AFTER MATCH SKIP PAST LAST ROW	If you use the clause, the next match starts to be performed from the event that follows the last event in the sequence of matched events.
AFTER MATCH SKIP TO FIRST patternItem	If you use the clause, the next match starts to be performed from the first event that corresponds to **patternItem** in the sequence of matched events.
AFTER MATCH SKIP TO LAST patternItem	If you use the clause, the next match starts to be performed from the last event that corresponds to **patternItem** in the sequence of matched events.
PATTERN	You can use the clause to specify the rule that the sequence of events to be identified must meet. You must enclose the rule in parentheses `()`. The rule is specified by a set of custom **patternVariable** variables. **Note** • If two **patternVariable** variables are separated with a space, no events exist between the events that correspond to the patternVariable variables. • If two **patternVariable** variables are separated with an arrow sign (**->**), other events may exist between the events that correspond to the **patternVariable** variables. • Blink V3.6.0 and later do not support the**PATTERN** clause due to Calcite upgrades.

Parameter description

- **quantifier**

  The **quantifier** parameter specifies the number of occurrences of events that meet the **patternVariable** definition.

Value	Description
*	Zero or multiple times.
+	One or more times.
?	Zero or once.
{n}	n times.
{n,}	At least n times.
{n, m}	Ranges from n times to m times.

{,m}	At most m times.

By default, greedy matching is performed. For example, if the PATTERN clause is `A -> B+ -> C` and the input is `a bc1 bc2 c`, the output is `a bc1 bc2 c`. In the input, bc1 and bc2 means that the results must match B and C. To perform reluctant matching, append the **quantifier** with a question mark (**?**). You can use the following reluctant quantifiers:

- `*?`
- `+?`
- `{n}?`
- `{n,}?`
- `{n, m}?`
- `{,m}?`

> **Note** Blink V3.x and later do not support **(e1 e2+)** greedy matching. You can use **e1 e2+ e3 e3 as not e2** as an alternative. In the alternative method, at least one e3 entry must be included to make sure that the output data is returned as expected.

In this case, the output that is generated for the input and the **PATTERN** setting in the preceding example changes to **a bc1 bc2,a bc1 bc2 c**.

- The **WITHIN** clause defines the maximum time span of the events that meet the specified rule in an event sequence.
- The format of static windows is `INTERVAL 'string' timeUnit [ TO timeUnit ]`, for 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`.
- The format of dynamic windows is `INTERVAL intervalExpression`, for example, `INTERVAL A.windowTime + 10`. In this example, A is the first **patternVariable** variable in the **PATTERN** clause. When you specify **intervalExpression**, you can use only the first **patternVariable** variable in the **PATTERN** clause. When you specify the intervalExpression parameter, you can use user defined functions (UDFs). The intervalExpression result indicates the window size that is measured in milliseconds. The data type of the result must be LONG.
- The DEFINE statement specifies the meanings of **patternVariable** variables in the **PATTERN** clause. If a **patternVariable** variable is not defined in the DEFINE statement, the **patternVariable** variable is valid for each event.

- Functions in **MEASURES and DEFINE statements**

Function	Description
Row Pattern Column References	This function uses the format of `patternVariable.col`. This function is used to access the specified column of an event that corresponds to a **patternVariable** variable.

PREV	This function can be used in only the DEFINE statement. In most cases, this function is used in conjunction with the `Row Pattern Column References` function. This function is used to access the specified column of a previous event that has a specified offset. The event occurs before the events that are specified by the **PATTERN** clause. For example, you can use `DOWN AS DOWN.price < PREV(DOWN.price)`. In this example, `PREV(A.price)` specifies the `price` column value for the event that occurs before the current event. **Note** ◦ `DOWN.price` is equivalent to `PREV(DOWN.price, 0)`. ◦ `PREV(DOWN.price)` is equivalent to `PREV(DOWN.price, 1)`.
FIRST or LAST	In most cases, the FIRST or the LAST function is used in conjunction with the `Row Pattern Column References` function. You can use the FIRST or the LAST function to access the event that has a specified offset in the sequence of events. The sequence of events is specified by the **PATTERN** clause. The following examples are used to explain the functions: ◦ `FIRST(A.price, 3)` specifies the fourth event in the sequence of events that match the `pattern A`. ◦ `LAST(A.price, 3)` specifies the last but three event in the sequence of events that match the `pattern A`.

- **Output columns**

Function	Output column
ONE ROW PER MATCH	The columns that are specified in `PARTITION BY` and `MEASURES` are included. The columns that are specified in `PARTITION BY` do not need to be specified in `MEASURES` again.
ONE ROW PER MATCH WITH TIMEOUT ROWS	An output event is generated for each match or time-out. The time-out period is specified by the **WITHIN** clause in **PATTERN**.

Note
- When you define the **PATTERN** clause, we recommend that you define the **WITHIN** clause. If the WITHIN clause is not defined, the state size may grow larger.
- The first column that is specified in the ORDER BY clause must be EVENT TIME or PROCESS TIME.

Examples

- Syntax in the example

```
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)
);
```

- Test data

timestamp (TIMESTAMP)	card_id(VARCHAR)	location (VARCHAR)	action (VARCHAR)
2018-04-13 12:00:00	1	Beijing	Consumption
2018-04-13 12:05:00	1	Shanghai	Consumption
2018-04-13 12:10:00	1	Shenzhen	Consumption
2018-04-13 12:20:00	1	Beijing	Consumption

- Syntax in the test
  Each credit card is identified by a unique card ID that is specified by the card_id parameter. If payments with a credit card are made within 10 minutes at two different locations, an alert is triggered. This helps you monitor unauthorized operations of credit cards.

```
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'
    ...
);

--Define the computational logic.
insert into rds_out
select
`start_timestamp`,
`end_timestamp`,
card_id, `event`
from datahub_stream
MATCH_RECOGNIZE (
    PARTITION BY card_id   --Partition the table by card ID. The data that has the same card ID is allocated to the same compute node.
    ORDER BY `timestamp`   --Sort events by time in a window.
    MEASURES               --Define how to generate output events based on the input events that are matched.
        e2.`action` as `event`,
        e1.`timestamp` as `start_timestamp`,   --Specify the time of the first event as the start_timestamp value.
        LAST(e2.`timestamp`) as `end_timestamp` --Specify the time of the last event as the end_timestamp value.
    ONE ROW PER MATCH           --The system generates an output event for each match.
    AFTER MATCH SKIP TO NEXT ROW --The system performs the next match from the next row after each match.
    PATTERN (e1 e2+) WITHIN INTERVAL '10' MINUTE  --Define two events: e1 and e2.
    DEFINE                     --Define the meanings of patternVariable variables in the PATTERN clause.
        e1 as e1.action = 'Consumption',    --Mark the action of the e1 event as Consumption.
        e2 as e2.action = 'Consumption' and e2.location <> e1.location --Mark the action of the e2 event as Consumption. The locations of e1 and e2 are different.
);
```

> **Note** In some scenarios, some data meets the CEP condition but no outputs are returned. This occurs because only data that meets the **watermark > e2.ts** condition is processed. No input data flows into the system after e2 and the watermark is always **e2.ts -1000**. As a result, the e2 data cannot be processed. Therefore, no outputs are returned.

- Test result

start_timestamp (TIMESTAMP)	end_timestamp (TIMESTAMP)	card_id (VARCHAR)	event (VARCHAR)
2018-04-13 12:00:00.0	2018-04-13 12:05:00.0	1	Consumption
2018-04-13 12:05:00.0	2018-04-13 12:10:00.0	1	Consumption

5.8.12. Deduplication statements

You can remove duplicates by executing statements such as FIRST_VALUE, LAST_VALUE, and DISTINCT. This topic describes how to execute TopN statements to remove duplicates and describes deduplication considerations.

You can remove duplicates by using the following methods:

- Keep the first row of duplicate rows.
- Keep the last row of duplicate rows.

> **Note** The time attribute column in the ORDER BY clause must be defined in the source table.

Syntax

Flink SQL does not support deduplication statements. You can execute the `ROW_NUMBER OVER WINDOW` statement of Flink SQL to remove duplicates. `ROW_NUMBER OVER WINDOW` is similar to a TopN statement and can be considered as a special TopN statement. For more information about TopN statements, see TopN.

```
SELECT *
FROM (
   SELECT *,
     ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]
     ORDER BY timeAttributeCol [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1;
```

The following list describes the parameters that are used in the preceding deduplication statement:

- `ROW_NUMBER()`: calculates the number of a row. The row number starts from 1.
- `PARTITION BY col1[, col2..]`: specifies one or more columns based on which partitioning is implemented. The specified partition key columns are used as the keys to remove duplicates. This parameter is optional.
- `ORDER BY timeAttributeCol [asc|desc])`: specifies the column that is used for sorting data. This column must be a time attribute column. The time attribute can be processing

time or event time. You can sort rows in ascending or descending order based on the time attribute. If you use the ascending order, the first row of duplicate rows is retained. If you use the descending order, the last row of duplicate rows is retained.

- The value of `rownum` in the outer query must be 1 (`= 1`) or greater than or equal to 1 (`<= 1`). In the outer query, the logical operator must be `AND` and you cannot use nondeterministic user-defined functions (UDFs).

Based on the preceding syntax, two levels of queries are required to remove duplicates:

- Subquery: calls the `ROW_NUMBER()` parameter that is used to sort data based on the time attribute column.
- Outer query: keeps the first row of duplicate rows that have the same primary key and removes the other duplicate rows. The following two sorting orders for the time attribute column are available:
  - Ascending: `deduplicate keep first row` .
  - Descending: `deduplicate keep last row` .

If the processing time column is used for sorting, Flink SQL removes duplicates based on the system time. In this case, the result may change each time the deduplication statement is executed. If the event time column is used for sorting, Flink SQL removes duplicates based on the business time. In this case, the result remains unchanged each time the deduplication statement is executed.

Deduplicate Keep First Row

If you select this policy, the system keeps the first row of duplicate rows that have the same primary key and discards the other duplicate rows. Only keys are stored in the state data of the job. This improves the job performance. You can use the following sample code to remove duplicates and keep the first row of duplicate rows:

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b ORDER BY proctime) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

In this example, the system removes duplicates in Table T based on Field b. The system keeps the first row of duplicate rows based on the system time. In this example, proctime is a field that has the processing time attribute. To remove deduplicates based on the system time, you can also call the `PROCTIME()` function instead of declaring the proctime filed.

> **Note** In a Blink version later than Blink V3.3.1, the deduplicate keep first row policy allows you to open windows by using the event time attribute. This operation does not trigger retraction.

Deduplicate Keep Last Row

> **Important** This policy does not allow you to open windows by using the event time attribute.

This policy is used to remove duplicates. The policy keeps only the last row of duplicate rows that have the same primary key. The performance of this policy is higher than that of the LAST_VALUE function. In the following sample code, the deduplicate keep last row policy is used to remove duplicates:

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY proctime DESC) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

FAQ

What do I do if the following error occurs when `ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY now() as time DESC)` is executed?

```
java.lang.RuntimeException: Can not retract a non-existent record:
    38c30001,1b800000008,1c000000013,85000035343a3731,5d304013.
    This should never happen.
```

This error may occur due to the following two causes:

- Cause: The `now()` function in the code may result in this error. The TopN function does not support nondeterministic sorting fields. The `now()` function is nondeterministic and returns a different value each time the function is invoked. Therefore, the previous value cannot be found during the retraction.

  Solution: Make sure that you use a deterministic time attribute field, such as a field of the event time attribute. You can also use a field of the processing time attribute in a source table as the deterministic time attribute field.
- Cause: The value of the `blink.state.ttl.ms` or `state.backend.niagara.ttl.ms` parameter is small.

  Solution: If the specified time-to-live (TTL) value is small, use the default value or increase the value.

5.9. Window functions

5.9.1. Overview

This topic describes the window functions, time attributes, and window types that Flink SQL supports.

Window functions

Flink SQL supports aggregation over infinite windows. You do not need to explicitly define windows in your SQL statements. Flink SQL also supports aggregation over a specific window. For example, to count the number of users who clicked a URL in the last minute, you can define a window to collect the data about user clicks that occur in the last minute. Then, you can compute the data in the window to obtain the result.

Flink SQL supports window aggregates and over aggregates. This topic describes window aggregates. Window aggregates support the windows that are defined based on the following two time attributes: event time and processing time. For each time attribute, Flink SQL supports three window types: tumbling window, sliding window, and session window.

Time attributes

Flink SQL supports two time attributes: event time and processing time. Realtime Compute for Apache Flink aggregates data in windows based on the following time attributes:

- Event time: the event time that you provide in the table schema. In most cases, the event time is the original time when data is created.
- Processing time: the local system time at which the system processes an event.

> **Note** For more information about the time attributes, see Time attributes.

Cascading windows

The event time attribute of the Rowtime column no longer takes effect after a window operation is complete. You can use a helper function such as `TUMBLE_ROWTIME`, `HOP_ROWTIME`, or `SESSION_ROWTIME` to obtain `max(rowtime)` of the rowtime column in a window. You can use the obtained value as the rowtime of the time window. The value is `window_end - 1` and is of the TIMESTAMP data type. The TIMESTAMP value has the rowtime attribute. For example, if the time span for a window is `[00:00, 00:15]`, `00:14:59.999` is returned.

In the following example, 1-hour tumbling windows are used to aggregate data based on the aggregation results of 1-minute tumbling windows. This helps you meet various window requirements.

```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000)  --Define a watermark for the rowtime.
) with (
  type='datahub',
  ...
);

CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='print'
);

CREATE VIEW one_minute_window_output as
SELECT
  // Use each TUMBLE_ROWTIME value as the aggregation time for the level-two window.
  TUMBLE_ROWTIME(ts, INTERVAL '1' MINUTE) as rowtime,
  username,
  COUNT(click_url) as cnt
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;

INSERT INTO tumble_output
SELECT
  TUMBLE_START(rowtime, INTERVAL '1' HOUR),
  TUMBLE_END(rowtime, INTERVAL '1' HOUR),
  username,
  SUM(cnt)
FROM one_minute_window_output
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), username;
```

5.9.2. TUMBLE

This topic describes how to use the TUMBLE function in Realtime Compute for Apache Flink.

Description

A TUMBLE function assigns each element to a tumbling window that has a specific size. In most cases, tumbling windows are fixed in size and do not overlap with each other. For example, if a 5-minute tumbling window is defined, an infinite data stream is divided into windows based on the time period, such as `[0:00, 0:05)`, `[0:05, 0:10)`, and `[0:10, 0:15)`.

Syntax

You can use a TUMBLE function in a GROUP BY clause to define a tumbling window.

```
TUMBLE(<time-attr>, <size-interval>)
<size-interval>: INTERVAL 'string' timeUnit
```

Note The `<time-attr>` parameter must be a valid time attribute field in a time stream. This parameter specifies whether the time is a processing time or an event time. For more information, see Time attributes, Watermark, and Overview.

Window identifier functions

A window identifier function specifies the start time, end time, or time attribute of a window. The time attribute is used to aggregate lower-level windows.

Function	Return value type	Description
`TUMBLE_START(time-attr, size-interval)`	TIMESTAMP	Returns the start time, including the boundary value, of a window. For example, if the time span of a window is `[00:10,00:15)`, `00:10` is returned.
`TUMBLE_END(time-attr, size-interval)`	TIMESTAMP	Returns the end time, including the boundary value, of a window. For example, if the time span of a window is `[00:00, 00:15)`, `00:15` is returned.
`TUMBLE_ROWTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	Returns the end time, excluding the boundary value, of a window. For example, if the time span of a window is `(00:00, 00:15)`, `00:14:59.999` is returned. The return value is a rowtime attribute value based on which time operations can be performed. For example, this function can be used in only the windows that are based on the event time, such as cascading windows. For more information, see Cascading window.
`TUMBLE_PROCTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	Returns the end time, excluding the boundary value, of a window. For example, if the time span of a window is `(00:00, 00:15)`, `00:14:59.999` is returned. The return value is a proctime attribute, based on which time operations can be performed. For example, a cascading window function can be used only in windows that are defined based on the processing time.

Example 1: Count the number of clicks per user per minute for a specific website based on the event time

- Test data

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`

- Test statements

```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000) --Define 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
TUMBLE_START(ts, INTERVAL '1' MINUTE) as window_start,
TUMBLE_END(ts, INTERVAL '1' MINUTE) as window_end,
username,
COUNT(click_url)
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
```

- Test results

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

Example 2: Count the number of clicks per user per minute for a specific website based on the processing time

- Test data

username (VARCHAR)	click_url (VARCHAR)
Jark	http://taobao.com/xxx
Jark	http://taobao.com/xxx
Jark	http://taobao.com/xxx
Jark	http://taobao.com/xxx
Jark	http://taobao.com/xxx
Timo	http://taobao.com/xxx

- Test statements

```
CREATE TABLE window_test (
  username   VARCHAR,
  click_url  VARCHAR,
  ts as PROCTIME()
) WITH (
  type='datahub',
  ...
);

CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) WITH (
  type='print'
);

INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE),
TUMBLE_END(ts, INTERVAL '1' MINUTE),
username,
COUNT(click_url)
FROM window_test
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
```

- Test results

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)

2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Jark	5
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Timo	1

Note Local debugging is instantaneous and the processing time may be less than 1s. Therefore, if the processing time attribute is used to aggregate data in windows, local debugging may fail.

5.9.3. HOP

This topic describes how to use the HOP function in Realtime Compute for Apache Flink.

Note In Realtime Compute for Apache Flink, the HOP function cannot be used in conjunction with LAST_VALUE, FIRST_VALUE, or TopN functions.

Introduction

A HOP function is used to define a hopping window, which is also known as a sliding window. Unlike tumbling windows, sliding windows can overlap with each other.

A sliding window is defined by the following parameters: **slide** and **size**. The **slide** parameter specifies the length of a sliding step. The **size** parameter specifies the size of the window.

- If the value of **slide** is less than the value of size, windows overlap with each other and each element is assigned to multiple windows.
- If the value of **slide** is equal to the value of size, windows are tumbling windows.
- If the value of **slide** is greater than the value of size, windows are sliding windows. These windows do not overlap with each other and are separated by gaps.

In most cases, most elements are assigned to multiple windows and the windows overlap with each other. Sliding windows are used to calculate moving averages. For example, to calculate the data average in the last 5 minutes every 10 seconds, set **slide** to 10 seconds and set **size** to 5 minutes.

Syntax

You can use the HOP function 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 must be a valid time attribute field in a stream. This parameter specifies whether the time is the processing time or the event time. For more information about Time attributes and Watermark, see Overview.

Window identifier functions

A window identifier function specifies the start time, end time, or time attribute of a window. The time attribute is used to aggregate lower-level windows.

Function	Return value type	Description
`HOP_START (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	Returns the start time, including the boundary value, of a window. For example, if the time span of a window is `[00:10, 00:15)`, `00:10` is returned.
`HOP_END (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	Returns the end time, including the boundary value, of a window. For example, if the time span of a window is `[00:00, 00:15)`, `00:15` is returned.
`HOP_ROWTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time, excluding the boundary value, of a window. For example, if the time span 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 operations can be performed. This function can be used in only the windows that are defined based on the event time, such as cascading windows. For more information, see Cascading windows.
`HOP_PROCTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time, excluding the boundary value, of a window. For example, if the time span of a window is `(00:00, 00:15)`, `00:14:59.999` is returned. The return value is a processing time attribute based on which time operations can be performed. This function can be used only in the windows that are defined based on the processing time, such as cascading windows. For more information, see Cascading windows.

Example

In the following example, a 1-minute window slides once every 30 seconds. You can use the windows to count the number of clicks per user over the last minute every 30 seconds.

- Test data

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`

- Test statements

```
CREATE TABLE user_clicks (
    username VARCHAR,
    click_url VARCHAR,
    ts TIMESTAMP,
    WATERMARK wk FOR ts AS WITHOFFSET (ts, 2000)--Define a watermark for rowtime.
) WITH ( TYPE = 'datahub',
        ...);

CREATE TABLE hop_output (
    window_start TIMESTAMP,
    window_end TIMESTAMP,
    username VARCHAR,
    clicks BIGINT
) WITH (TYPE = 'rds',
        ...);

INSERT INTO
    hop_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
    user_clicks
GROUP BY
    HOP (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    username
```

- Test results

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 09:59:30.0	2017-10-10 10:00:30.0	Jark	2
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 10:02:00.0	2017-10-10 10:03:00.0	Timo	1
2017-10-10 10:02:30.0	2017-10-10 10:03:30.0	Timo	1

If a sliding window cannot read the time at which data enters the window, the start time of the first window is moved forward. You can use the following formula to calculate the time interval by which the start time is moved forward: **Time interval = Window duration - Sliding step**.

Window duration (seconds)	Sliding step (seconds)	Event Time	Start time of the first window	End time of the first window
120	30	`2019-07-31 10:00:00.0`	`2019-07-31 09:58:30.0`	`2019-07-31 10:00:30.0`
60	10	`2019-07-31 10:00:00.0`	`2019-07-31 09:59:10.0`	`2019-07-31 10:00:10.0`

5.9.4. SESSION

This topic describes how to use the SESSION function in Realtime Compute for Apache Flink.

Introduction

A SESSION function groups elements by session activity. Unlike tumbling and sliding windows, session windows do not overlap and are not fixed in size. If a session window does not receive elements within a specific period of time, the session is disconnected and the window is closed.

A session window is configured by using a gap, which defines the length of the inactive period. For example, a data stream that represents mouse click activities may include highly clustered mouse click events, separated by inactive periods. Data that arrives after a specified gap is assigned to a new window.

Syntax

You can use a SESSION function in a GROUP BY clause to define a session window.

```
SESSION(<time-attr>, <gap-interval>)
<gap-interval>: INTERVAL 'string' timeUnit
```

> **Note** The `<time-attr>` parameter must be a valid time attribute in a data stream to specify whether the time is the processing time or event time. For more information, see Overview, Time attributes, and Watermark.

Window identifier functions

A window identifier function specifies the start time, end time, or time attribute of a window. The time attribute is used to aggregate lower-level windows.

Function	Return value type	Description
`SESSION_START(<time-attr>, <gap-interval>)`	Timestamp	Returns the start time, including the boundary value, of a window. For example, if the time span of a window is `[00:10, 00:15)`, `00:10` is returned. The return value is the time of the first record in the session window.

`SESSION_END (<time-attr>, <gap-interval>)`	Timestamp	Returns the end time, including the boundary value, of a window. For example, if the time span of a window is `[00:00, 00:15]`, `00:15` is returned. The return value is the time of the last record in the session window plus `<gap-interval>`.
`SESSION_ROWTIME (<time-attr>, <gap-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time, excluding the boundary value, of a window. For example, if the time span of a window is `(00:00, 00:15)`, `00:14:59.999` is returned. The return value is an event time attribute, based on which time type operations such as window cascading can be performed. The function applies only to windows based on the event time.
`SESSION_PROCTIME (<time-attr>, <gap-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time, excluding the boundary value, of a window. For example, if the time span of a window is `(00:00, 00:15)`, `00:14:59.999` is returned. The return value is a processing time attribute, based on which time type operations such as window cascading can be performed. The function applies only to windows based on the processing time.

Example

The following example describes how to calculate the number of clicks per user during each active session. The session timeout interval is 30 seconds.

- Test data

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`

- Test statements

```
CREATE TABLE user_clicks(
username varchar,
click_url varchar,
ts timeStamp,
WATERMARK wk FOR ts as withOffset(ts, 2000) -- Define a watermark for rowtime.
) WITH (
type='datahub',
...
);

CREATE TABLE session_output(
window_start TIMESTAMP,
window_end TIMESTAMP,
username VARCHAR,
clicks BIGINT
) WITH (
type='rds',
...
);

INSERT INTO session_output
SELECT
SESSION_START(ts, INTERVAL '30' SECOND),
SESSION_END(ts, INTERVAL '30' SECOND),
username,
COUNT(click_url)
FROM user_clicks
GROUP BY SESSION(ts, INTERVAL '30' SECOND), username;
```

- Test results

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 10:01:35.0	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

5.9.5. OVER windows

An OVER window is a standard window used in traditional databases. Over aggregate is different from window aggregate. In streaming data that uses OVER windows, each element corresponds to an OVER window. An OVER window can be determined based on an actual row or an actual value (timestamp value) of an element. Elements of a stream are distributed across multiple windows.

In a stream that applies the OVER window, each element corresponds to an OVER window and triggers data computing once. The row determined by each element that triggers computing is the last row of the window where the element is located. In the underlying implementation of Realtime Compute, the OVER window data is centrally managed. Only one copy of the data is stored. Logically, an OVER window is created for each element. Realtime Compute for Apache Flink calculates the data for each OVER window and then deletes the data that is no longer used after the calculation is complete. For more information, see Over Aggregation.

Syntax

```
SELECT
    agg1(col1) OVER (definition1) AS colName,
    ...
    aggN(colN) OVER (definition1) AS colNameN
FROM Tab1;
```

- agg1(col1): aggregates input data based on the col1 column specified by GROUP BY.
- OVER (definition1): defines an OVER window.
- AS colName: specifies the alias of a column.

> **Note**
> - OVER (definition1) for agg1 through aggN must be the same.
> - The alias specified by AS can be queried by using an outer SQL statement.

Window types

In Flink SQL, OVER windows are defined in compliance with standard SQL syntax. The traditional OVER windows are not classified into fine-grained window types. OVER windows are classified into the following two types based on the ways of determining computed rows:

- ROWS OVER window: Each row of elements is treated as a new computed row. A new window is generated for each row.
- RANGE OVER window: All rows of elements with the same timestamp value are treated as one computed row and are assigned to the same window.

Attributes

Orthogonal attribute	Description	proctime	eventtime
ROWS OVER Window	A window is determined based on the actual row of an element.	Supported	Supported
RANGE OVER Window	A window is determined based on the timestamp value of an element.	Supported	Supported

ROWS OVER window

- Description

  For a ROWS OVER window, a window is generated for each element.

- 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 to sort elements.
  - rowCount: specifies the number of rows that precede the current row.

- Example

  This example describes bounded ROWS OVER windows. In this example, an on-sale product table contains item IDs, item types, launch time, and prices. Calculate the highest price among the three products similar to the current product before the current product is on sale.

- Test data

Item ID	Item type	On-sale time	Price
ITEM001	Electronic	`2017-11-11 10:01:00`	20
ITEM002	Electronic	`2017-11-11 10:02:00`	50
ITEM003	Electronic	`2017-11-11 10:03:00`	30
ITEM004	Electronic	`2017-11-11 10:03:00`	60
ITEM005	Electronic	`2017-11-11 10:05:00`	40
ITEM006	Electronic	`2017-11-11 10:06:00`	20
ITEM007	Electronic	`2017-11-11 10:07:00`	70
ITEM008	Clothes	`2017-11-11 10:08:00`	20

- Test statements

```
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;
```

- Test results

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	60
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

RANGE OVER windows

- Description

  For a RANGE OVER window, all elements with the same timestamp value are assigned to the same window.

- 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 to sort elements.
- timeInterval: specifies the time interval between the time of the current row and that of the element row to which it can be traced back.

- Example

  This example describes bounded RANGE OVER windows. In this example, an on-sale product table contains item IDs, item types, launch time, and prices. Calculate the highest price among similar products that are on sale two minutes earlier than the current product.

  - Test data

Item ID	Item type	On-sale time	Price
ITEM001	Electronic	`2017-11-11 10:01:00`	20
ITEM002	Electronic	`2017-11-11 10:02:00`	50
ITEM003	Electronic	`2017-11-11 10:03:00`	30
ITEM004	Electronic	`2017-11-11 10:03:00`	60
ITEM005	Electronic	`2017-11-11 10:05:00`	40
ITEM006	Electronic	`2017-11-11 10:06:00`	20
ITEM007	Electronic	`2017-11-11 10:07:00`	70
ITEM008	Clothes	`2017-11-11 10:08:00`	20

- Test statements

```
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;
```

- Test results

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	40
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

5.10. Built-in functions

5.10.1. String functions

5.10.1.1. REGEXP_EXTRACT

This topic describes how to use the string function REGEXP_EXTRACT in Realtime Compute.

Syntax

```
VARCHAR REGEXP_EXTRACT(VARCHAR str, VARCHAR pattern, INT index)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string.
pattern	VARCHAR	The regular expression pattern.
index	INT	The index number of the substring to be extracted from the source string.

Important Comply with Java code conventions to write regular expression constants. When you run the codegen tool, it automatically converts SQL constant strings to Java code. Write the string \d as '\d' in the regular expression, just in the same way as you write a regular expression in Java.

Function description

This function extracts the substring with the specified index number from a string based on the specified regular expression pattern. The index number starts from 1. If any input parameter is NULL or the regular expression is invalid, the return value is NULL.

Examples

- Test data

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
foothebar	foo(. *?)(bar)	2
100-200	(\\d+)-(\\d+)	1
null	foo(. *?)(bar)	2
foothebar	null	2
foothebar	Empty string	2
foothebar	(	2

- Test statements

```
SELECT REGEXP_EXTRACT(str1, pattern1, index1) as result
FROM T1
```

- Test results

result(VARCHAR)
bar
100
null
null
null

null

5.10.1.2. REGEXP_REPLACE

This topic describes how to use the string function REGEXP_REPLACE in Realtime Compute for Apache Flink.

Syntax

```
VARCHAR REGEXP_REPLACE(VARCHAR str, VARCHAR pattern, VARCHAR replacement)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The string.
pattern	VARCHAR	The substring to be replaced in the source string.
replacement	VARCHAR	The substring that is used to replace the substring that matches the regular expression.

> **Important** The constants in the regular expression must comply with Java code standards. Codegen converts SQL string constants to Java code. Write the string `(\d)` as `'\d'` in the regular expression, which is in the same way you write a regular expression in Java.

Description

Replaces a substring that matches a specified regular expression pattern in the source string with another substring, and returns a new string. If any input parameter is null or the regular expression is invalid, the return value is null.

Example

- Test data

str1(VARCHAR)	pattern1(VARCHAR)	replace1(VARCHAR)
2014-03-13	-	Empty string
NULL	-	Empty string
2014-03-13	-	null
2014-03-13	Empty string	s
2014-03-13	(	s
100-200	(\d+)	num

- Test statements

```
SELECT  REGEXP_REPLACE(str1, pattern1, replace1) as result
FROM T1;
```

- Test results

result(VARCHAR)
20140313
null
null
2014-03-13
null
num-num

5.10.1.3. REPEAT

This topic describes how to use the string function REPEAT in Realtime Compute.

Syntax

```
VARCHAR REPEAT(VARCHAR str, INT n)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The string to be repeated.
n	INT	The number of times to repeat the string.

Function description

This function repeats a string a specified number of times and returns a new string. If str is NULL, the return value is NULL. If n is 0 or negative, the return value is an empty string.

Examples

- Test data

str(VARCHAR)	n(INT)
J	9
Hello	2
Hello	-9
null	9

- Test statements

```
SELECT REPEAT(str,n) as var1
FROM T1
```

- Test results

var1(VARCHAR)
JJJJJJJJ
HelloHello
Empty string
null

5.10.1.4. REPLACE

This topic describes how to use the string function REPLACE in Realtime Compute.

Syntax

```
VARCHAR REPLACE(str1, str2, str3)
```

Input parameters

Parameter	Data type	Description
str1	VARCHAR	The source string.
str2	VARCHAR	The substring to be replaced in the source string.
str3	VARCHAR	The replacement substring.

Function description

This function replaces a substring of a string with another substring.

Examples

- Test data

str1(INT)	str2(INT)	str3(INT)
alibaba blink	blink	flink

- Test statements

```
SELECT REPLACE(str1, str2, str3) as `result`
FROM T1
```

- Test results

result(VARCHAR)
alibaba flink

5.10.1.5. REVERSE

This topic describes how to use the string function REVERSE in Realtime Compute.

Syntax

```
VARCHAR REVERSE(VARCHAR str)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The string.

Function description

This function returns a string in the reverse order of the specified string. If any input parameter is NULL, the return value is NULL.

Examples

- Test data

str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)	str4(VARCHAR)
iPhoneX	Alibaba	World	null

- Test statements

```
SELECT REVERSE(str1) as var1,REVERSE(str2) as var2,
        REVERSE(str3) as var3,REVERSE(str4) as var4
FROM T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)
XenohPi	ababilA	dlroW	null

5.10.1.6. RPAD

This topic describes how to use the string function RPAD in Realtime Compute.

Syntax

```
VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
```

Input parameters

Parameter	Data type	Description

str	VARCHAR	The source string.
len	INT	The length of the new string after padding.
pad	VARCHAR	The string to be repeatedly padded to the source string.

Function description

This function right-pads a source string with another string several times until the new string reaches the specified length. If any input parameter is NULL, the return value is NULL.

If `len` is negative, the return value is NULL.

If `pad` is an empty string and the value of `len` is less than or equal to the length of `str`, `str` is trimmed to the specified length. If pad is an empty string and the value of `len` is greater than the length of `str`, the return value is NULL.

Examples

- Test data

str(VARCHAR)	len(INT)	pad(VARCHAR)
Empty string	-2	Empty string
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	Empty string
Empty string	2	s
asd	4	Empty string
Empty string	0	Empty string

- Test statements

```
SELECT RPAD(str, len, pad) as result
FROM T1
```

- Test results

result(VARCHAR)
null
HelloWorldJohnJ
Jo

CHel
null
null
as
ss
null
Empty string

5.10.1.7. SPLIT_INDEX

This topic describes how to use the string function SPLIT_INDEX in Realtime Compute.

Syntax

```
VARCHAR SPLIT_INDEX(VARCHAR str, VARCHAR sep, INT index)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string to be split.
sep	VARCHAR	The separator.
index	INT	The index number of the substring to be extracted from the source string.

Function description

This function uses the separator specified by `sep` to split the string specified by `str` into several substrings and returns the substring indexed as `index`. The value of `index` starts from 0. If the substring with the specified index number does not exist, the return value is NULL.

If any input parameter is NULL, the return value is NULL.

Examples

- Test data

str(VARCHAR)	sep(VARCHAR)	index(INT)
Jack,John,Mary	,	2
Jack,John,Mary	,	3
Jack,John,Mary	null	0
null	,	0

- Test statements

```
SELECT SPLIT_INDEX(str, sep, index) as var1
FROM T1
```

- Test results

var1(VARCHAR)
Mary
null
null
null

5.10.1.8. STR_TO_MAP

This topic describes how to use the string function STR_TO_MAP in Realtime Compute.

Syntax

```
MAP STR_TO_MAP(VARCHAR text)
MAP STR_TO_MAP(VARCHAR text, VARCHAR listDelimiter, VARCHAR keyValueDelimiter)
```

Function description

This function first uses the separator specified by listDelimiter to split the given text into key-value pairs. Then, this function uses the separator specified by keyValueDelimiter to separate the key and value in each key-value pair. Finally, this function assembles and returns a MAP. The default value of listDelimiter is a comma (,). The default value of keyValueDelimiter is an equal sign (=).

Input parameters

Parameter	Data type	Description
text	VARCHAR	The input text.
listDelimiter	VARCHAR	The separator between key-value pairs in the input text. The default value is a comma (,).
keyValueDelimiter	VARCHAR	The separator between the key and value in each key-value pair. The default value is an equal sign (=).

> **Important** The listDelimiter and keyValueDelimiter parameters are defined by Java regular expressions. If a special character is used, it needs to be escaped.

Test statements

```
SELECT
  STR_TO_MAP('k1=v1,k2=v2')['k1'] as a
FROM T1
```

Test results

a(VARCHAR)
v1

5.10.1.9. SUBSTRING

This topic describes how to use the string function SUBSTRING in Realtime Compute.

Syntax

```
VARCHAR SUBSTRING(VARCHAR a, INT start)
VARCHAR SUBSTRING(VARCHAR a, INT start, INT len)
```

Input parameters

Parameter	Data type	Description
a	VARCHAR	The source string.
start	INT	The start position of the substring to be extracted from the source string.
len	INT	The length of the substring to be extracted.

Function description

This function returns a substring of the specified length from a string, starting from the specified position. If the length is not specified, this function returns the substring from the specified position to the end of the string. The value of start begins with 1. If the value is 0, it is regarded as 1. If the value is negative, this function counts backward from the end of the string to find the first character of the substring.

Examples

- Test data

str(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	null

- Test statements

```
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
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)
Empty string	1=v1;k2=v2	v2	v	1	Empty string	Empty string	k1=v1;k2=v2	k1=v1;k2=v2	null

5.10.1.10. TO_BASE64

This topic describes how to use the string function TO_BASE64 in Realtime Compute.

Syntax

```
VARCHAR TO_BASE64(bin)
```

Input parameters

Parameter	Data type
bin	BINARY

Function description

This function converts binary data to a Base64-encoded string.

Examples

- Test data

c(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

- Test statements

```
SELECT TO_BASE64(FROM_BASE64(c)) as var1
FROM T1
```

- Test results

var1(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

5.10.1.11. TRIM

This topic describes how to use the string function TRIM in Realtime Compute.

Syntax

```
VARCHAR TRIM( VARCHAR x )
```

Input parameters

Parameter	Data type
x	VARCHAR

Function description

This function removes leading and trailing characters from a string. The most common use is to remove leading and trailing spaces.

Examples

- Test statements

```
SELECT TRIM('   Sample   ') as result
FROM T1
```

- Test results

result(VARCHAR)
Sample

> **Note** The return value is 'Sample'.

5.10.1.12. UPPER

This topic describes how to use the string function UPPER in Realtime Compute.

Syntax

```
VARCHAR UPPER(A)
```

Input parameters

Parameter	Data type
A	VARCHAR

Function description

This function converts all the letters in a string to uppercase.

Examples

- Test data

var1(VARCHAR)
ss
ttee

- Test statements

```
SELECT UPPER(var1) as aa
FROM T1;
```

- Test results

aa(VARCHAR)
SS
TTEE

5.10.1.13. CHAR_LENGTH

This topic describes how to use the string function CHAR_LENGTH in Realtime Compute.

Syntax

```
INT CHAR_LENGTH(A)
```

Input parameters

Parameter	Data type
A	INT

Function description

This function returns the number of characters contained in a string.

Examples

- Test data

var1(INT)
ss
231ee

- Test statements

```
SELECT CHAR_LENGTH(var1) as aa
FROM T1;
```

- Test results

aa(INT)
2
5

5.10.1.14. CHR

This topic describes how to use the string function CHR in Realtime Compute.

Syntax

```
VARCHAR CHR(INT ascii)
```

Input parameters

Parameter	Data type	Description
ascii	INT	The integer ranging from 0 to 255. If the input parameter falls out of this range, the return value is NULL.

Function description

This function converts an ASCII code into a character.

Examples

- Test data

int1(INT)	int2(INT)	int3(INT)
255	97	65

- Test statements

```
SELECT CHR(int1) as var1, CHR(int2) as var2, CHR(int3) as var3
FROM T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
ÿ	a	A

5.10.1.15. CONCAT

This topic describes how to use the string function CONCAT in Realtime Compute.

Syntax

```
VARCHAR CONCAT(VARCHAR var1, VARCHAR var2, ...)
```

Input parameters

Parameter	Data type	Description
var1	VARCHAR	The string.
var2	VARCHAR	The string.

Function description

This function concatenates two or more strings into a single string. If any input parameter is NULL, the parameter is skipped.

Examples

- Test data

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
Hello	My	World
Hello	null	World
null	null	World
null	null	null

- Test statements

```
SELECT CONCAT(var1, var2, var3) as var
FROM T1
```

- Test results

var(VARCHAR)
HelloMyWorld
HelloWorld
World
null

5.10.1.16. CONCAT_WS

This topic describes how to use the string function CONCAT_WS in Realtime Compute.

Syntax

```
VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
```

Input parameters

Parameter	Data type	Description
separator	VARCHAR	The separator.
var1	VARCHAR	The parameter to be concatenated.
var2	VARCHAR	The parameter to be concatenated.

Function description

This function concatenates every two parameter values with a separator and returns a new string. The length and type of the new string depend on the input values.

> **Note** If the separator value is NULL, it is regarded as an empty string for concatenating the parameter values. If any other parameter is NULL, the parameter is skipped during concatenation.

Examples

- Test data

sep(VARCHAR)	str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)
|	Jack	Harry	John
null	Jack	Harry	John
|	null	Harry	John
|	Jack	null	null

- Test statements

```
SELECT CONCAT_WS(sep, str1, str2, str3) as var
FROM T1
```

- Test results

var(VARCHAR)
Jack|Harry|John
JackHarryJohn

Harry|John
Jack

5.10.1.17. FROM_BASE64

This topic describes how to use the string function FROM_BASE64 in Realtime Compute.

Syntax

```
BINARY FROM_BASE64(str)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The Base64-encoded string.

Function description

This function decodes a Base64-encoded string into binary data.

Examples

- Test data

a(INT)	b(BIGINT)	c(VARCHAR)
1	1L	null

- Test statements

```
SELECT
from_base64(c) as var1,from_base64('SGVsbG8gd29ybGQ=') as var2
FROM T1
```

- Test results

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')]

5.10.1.18. HASH_CODE

This topic describes how to use the string function HASH_CODE in Realtime Compute.

Syntax

```
INT HASH_CODE(VARCHAR str)
```

Input parameters

str	VARCHAR

Function description

This function generates a hash code for the specified string based on the HASH_CODE() method, and then returns the absolute value of the hash code.

Examples

- Test data

str1(VARCHAR)	str2(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	k1:v1,k2:v2	null

- Test statements

```
SELECT HASH_CODE(str1) as var1, HASH_CODE(str2) as var2, HASH_CODE(nullstr) as var3
FROM T1
```

- Test results

var1(INT)	var2(INT)	var3(INT)
1099348823	401392878	null

5.10.1.19. INITCAP

This topic describes how to use the string function INITCAP in Realtime Compute.

Syntax

```
VARCHAR INITCAP(A)
```

Input parameters

Parameter	Data type
A	VARCHAR

Description

Returns a string in which the first letter of each word is uppercase and all other letters are lowercase.

Example

- Test data

var1 (VARCHAR)
aADvbn

- Test statements

```
SELECT INITCAP(var1)as aa
FROM T1;
```

- Test results

aa (VARCHAR)
Aasvbn

5.10.1.20. INSTR

This topic describes how to use the string function INSTR in Realtime Compute.

> **Note** The INSTR function is available only in Realtime Compute V2.2.0 and later.

Syntax

```
INT instr( string1, string2 )
INT instr( string1, string2 [, start_position [, nth_appearance ] ] )
```

Input parameters

Parameter	Data type	Description
string1	VARCHAR	The source string to search.
string2	VARCHAR	The substring to search for in the source string.
start_position	INT	The position in the source string where the search starts.
nth_appearance	INT	Which occurrence of the substring to be searched for in the source string.

Function description

This function returns the position of the substring in the source string. If the substring is not found in the source string, the return value is 0.

Examples

- Test data T1

string1(VARCHAR)
helloworld

- Test statements

```
SELECT
instr('helloworld','lo') as res1,
instr('helloworld','l',-1,1) as res2,
instr('helloworld','l',3,2) as res3
FROM T1
```

- Test results

res1(INT)	res2(INT)	res3(INT)
4	9	4

5.10.1.21. JSON_VALUE

This topic describes how to use the string function JSON_VALUE in Realtime Compute for Apache Flink.

Syntax

```
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
```

Input parameters

- content

  The JSON object that you want to parse, which is represented as a string. This parameter is of the VARCHAR type.

- path
  The path expression that is used to parse the JSON object. This parameter is of the VARCHAR type. The following table describes the path expressions that are supported by the JSON_VALUE function.

Symbol	Description
$	The root object.
[]	The array subscript.
*	The array wildcard.
.	The child element.

Description

Extracts the value of the specific path from a JSON string. If the JSON string is invalid or an input parameter is null, null is returned.

Example

- Test data

id(INT)	json(VARCHAR)	path1(VARCHAR)
1	[10, 20, [30, 40]]	$[2][*]
2	null	$.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][*]"

- Test statement

```
SELECT
    id,
    JSON_VALUE(json, path1) AS `value`
FROM
    T1;
```

- Test results

id (INT)	value (VARCHAR)
1	[30,40]
2	["h0","h1","h2"]
3	h1
4	NULL
5	NULL
6	NULL

5.10.1.22. KEYVALUE

This topic describes how to use the string function KEYVALUE in Realtime Compute.

Syntax

```
VARCHAR KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The key-value pairs in the specified string.
split1	VARCHAR	The separator between key-value pairs.
split2	VARCHAR	The separator between the key and value in each key-value pair.

key_name	VARCHAR	The name of the key whose value is to be extracted.

Function description

This function parses the key-value pairs in a string based on the key-value pair separator and key-value separator. Then, this function returns the value for the specified key name. If the key name does not exist or an exception occurs, the return value is NULL.

Examples

- Test data

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	|	=	:

- Test statements

```
SELECT KEYVALUE(str, split1, split2, key1) as `result`
FROM T1
```

- Test results

result(VARCHAR)
v2
null
null
null
null
null

5.10.1.23. LOWER

This topic describes how to use the string function LOWER in Realtime Compute.

Syntax

```
VARCHAR LOWER(A)
```

Input parameters

- A

  VARCHAR

Function description

This function converts all the letters in a string to lowercase.

Examples

- Test data

var1(VARCHAR)
Ss
yyT

- Test statements

```
SELECT LOWER(var1) as aa
FROM T1;
```

- Test results

aa(VARCHAR)
ss
yyt

5.10.1.24. LPAD

This topic describes how to use the string function LPAD in Realtime Compute.

Syntax

```
VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The source string.
len	INT	The length of the new string after padding.
pad	VARCHAR	The string to be repeatedly padded to the source string.

Function description

This function left-pads a source string with another string several times until the new string reaches the specified length.

If any input parameter is NULL, the return value is NULL.

If `len` is negative, the return value is NULL.

If `pad` is an empty string and the value of `len` is less than or equal to the length of `str`, `str` is trimmed to the specified length. If pad is an empty string and the value of `len` is greater than the length of `str`, the return value is NULL.

Examples

- Test data

str(VARCHAR)	len(INT)	pad(VARCHAR)
Empty string	-2	Empty string
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	Empty string
Empty string	2	s
asd	4	Empty string
Empty string	0	Empty string

- Test statements

```
SELECT LPAD(str, len, pad) AS result
FROM T1
```

- Test results

result(VARCHAR)
null
JohnJHelloWorld
Jo
HelC
null
null
as
ss

null
Empty string

5.10.1.25. MD5

This topic describes how to use the string function MD5 in Realtime Compute.

Syntax

```
VARCHAR MD5(VARCHAR str)
```

Input parameters

- str

  VARCHAR

Function description

This function returns the MD5 value of the specified string. If the input parameter is an empty string (''), the return value is an empty string.

Examples

- Test data

str1(VARCHAR)	str2(VARCHAR)
k1=v1;k2=v2	Empty string

- Test statements

```
SELECT
   MD5(str1) as var1,
   MD5(str2) as var2
FROM T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)
19c17f42b4d6a90f7f9ffc2ea9bdd775	Empty string

5.10.1.26. OVERLAY

This topic describes how to use the string function OVERLAY in Realtime Compute for Apache Flink.

Syntax

```
VARCHAR OVERLAY ( (VARCHAR x PLACING VARCHAR y FROM INT start_position [ FOR INT length ]) )
```

Input parameters

Parameter	Data type
x	VARCHAR
y	VARCHAR
start_position	INT
length (optional)	INT

Description

Replaces the substring of x with y. The replacement starts from the position specified by start_position. The total number of characters to be replaced is the length value plus one.

Example

- Test statements

```
OVERLAY('abcdefg' PLACING 'hij' FROM 2 FOR 2) as result
FROM  T1;
```

- Test results

result(VARCHAR)
ahijdefg

5.10.1.27. PARSE_URL

This topic describes how to use the string function PARSE_URL in Realtime Compute.

Syntax

```
VARCHAR PARSE_URL(VARCHAR urlStr, VARCHAR partToExtract [, VARCHAR key])
```

Input parameters

Parameter	Data type	Description
urlStr	VARCHAR	The URL string.
partToExtract	VARCHAR	The part to be parsed from the URL.
key (optional)	VARCHAR	The name of the key whose value is to be extracted.

Function description

This function parses a URL and returns the specified part from the URL. If the value of partToExtract is QUERY, this function returns the value of the specified key in the URL. Valid values of partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, FILE, AUTHORITY, and USERINFO.

> **Important** If the input URL string is NULL, the return value is NULL.

Examples

- Test data

url1(VARCHAR)	nullstr(VARCHAR)
http://facebook.com/path/p1.php?query=1	null

- Test statements

```
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
```

- Test results

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

5.10.1.28. POSITION

This topic describes how to use the string function POSITION in Realtime Compute for Apache Flink.

Syntax

```
INTEGER POSITION( x IN  y)
```

Input parameters

Parameter	Data type
x	VARCHAR
y	VARCHAR

Description

Returns the position where the substring x appears for the first time in the source string y. If the substring does not exist in the source string, the return value is 0.

Example

- Test statements

```
POSITION('in' IN 'china') as result
FROM T1;
```

- Test results

result(INT)
3

5.10.1.29. REGEXP

This topic describes how to use the string function REGEXP in Realtime Compute for Apache Flink.

Syntax

```
BOOLEAN REGEXP(VARCHAR str, VARCHAR pattern)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The string.
pattern	VARCHAR	The regular expression pattern.

Description

Performs regular expression matching on a specified string to check whether it matches a specified pattern. If the string or pattern is empty or null, the return value is false.

Example

- Test data

str1(VARCHAR)	pattern1(VARCHAR)
k1=v1;k2=v2	k2*
k1:v1|k2:v2	k3
null	k3
k1:v1|k2:v2	null
k1:v1|k2:v2	(

- Test statements

```
SELECT  REGEXP(str1, pattern1) AS result
FROM T1;
```

- Test results

result(BOOLEAN)
true
false
false
false
false

5.10.2. Mathematical functions

5.10.2.1. Addition

This topic describes how to use the mathematical function addition in Realtime Compute.

Syntax

```
A + B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns the sum of A and B.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
10	20	30

- Test statements

```
SELECT int1+int2+int3 as aa
FROM T1;
```

- Test results

aa (INT)

60

5.10.2.2. Subtraction

This topic describes how to use the mathematical function subtraction in Realtime Compute.

Syntax

```
A - B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns the result of A minus B.

Examples

- Test data

int1(INT)	int2(INT)	int3(INT)
10	10	30

- Test statements

```
SELECT int3 - int2 - int1 as aa
FROM T1
```

- Test results

aa(int)
10

5.10.2.3. Multiplication

This topic describes how to use the mathematical function multiplication in Realtime Compute.

Syntax

```
A * B
```

Input parameters

Parameter	Data type

A	INT
B	INT

Function description

This function returns the result of A multiplied by B.

Examples

- Test data

int1(INT)	int2(INT)	int3(INT)
10	20	3

- Test statements

```
SELECT int1*int2*int3 as aa
FROM T1
```

- Test results

aa(int)
600

5.10.2.4. Division

This topic describes how to use the mathematical function division in Realtime Compute.

Syntax

```
A/B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns the result of A divided by B.

Examples

- Test data

int1(INT)	int2(INT)
8	4

- Test statements

```
SELECT int1/int2 as aa
FROM T1
```

- Test results

aa(int)
2

5.10.2.5. ABS

This topic describes how to use the mathematical function ABS in Realtime Compute.

Syntax

```
DOUBLE ABS(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the absolute value of input parameter A.

Examples

- Test data

in1(DOUBLE)
4.3

- Test statements

```
SELECT ABS(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
4.3

5.10.2.6. ACOS

This topic describes how to use the mathematical function ACOS in Realtime Compute.

Syntax

```
ACOS(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the arccosine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.7173560908995228
0.4

- Test statements

```
SELECT ACOS(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
0.7707963267948966
1.1592794807274085

5.10.2.7. BIN

This topic describes how to use the mathematical function BIN in Realtime Compute.

Syntax

```
VARCHAR BIN(BIGINT number)
```

Input parameters

Parameter	Data type
number	BIGINT **Note** You can set this parameter only to a BIGINT or INT value. The INT value is implicitly converted to a BIGINT value for computation.

Function description

This function converts a BIGINT value to a binary string.

Examples

- Test data

id(INT)	x(BIGINT)
1	12L
2	10L
3	0L
4	10000000000L

> **Note** In the test data, letter `L` in the **x(BIGINT)** column indicates the data type **Long**, which is not involved in binary conversion.

- Test statements

```
SELECT id, bin(x) as var1
FROM T1
```

- Test results

id(INT)	var1(VARCHAR)
1	1100
2	1010
3	0
4	1001010100000010111110010000000000

5.10.2.8. ASIN

This topic describes how to use the mathematical function ASIN in Realtime Compute.

Syntax

```
DOUBLE ASIN(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the arcsine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.7173560908995228
0.4

- Test statements

```
SELECT ASIN(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
0.8
0.41151684606748806

5.10.2.9. ATAN

This topic describes how to use the mathematical function ATAN in Realtime Compute.

Syntax

```
DOUBLE ATAN(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the arctangent value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.7173560908995228
0.4

- Test statements

```
SELECT ATAN(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
0.6222796222326533
0.3805063771123649

5.10.2.10. BITAND

This topic describes how to use the mathematical function BITAND in Realtime Compute.

Syntax

```
INT BITAND(INT number1, INT number2)
```

Input parameters

Parameter	Data type
number1	INT
number2	INT

Function description

This function performs a bitwise AND operation on the specified values. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)	b(INT)
2	3

- Test statements

```
SELECT BITAND(a, b) as intt
FROM T1
```

- Test results

intt(INT)
2

5.10.2.11. BITNOT

This topic describes how to use the mathematical function BITNOT in Realtime Compute.

Syntax

```
INT BITNOT(INT number)
```

Input parameters

Parameter	Data type
number	INT

Function description

This function performs a bitwise NOT operation on the specified value. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)
7

- Test statements

```
SELECT BITNOT(a) as var1
FROM T1
```

- Test results

var1(INT)
0xfff8

5.10.2.12. BITOR

This topic describes how to use the mathematical function BITOR in Realtime Compute.

Syntax

```
INT BITOR(INT number1, INT number2)
```

Input parameters

Parameter	Data type
number1	INT
number2	INT

Function description

This function performs a bitwise OR operation on the specified values. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)	b(INT)
2	3

- Test statements

```
SELECT BITOR(a, b) as var1
FROM T1
```

- Test results

var1(INT)
3

5.10.2.13. BITXOR

This topic describes how to use the mathematical function BITXOR in Realtime Compute.

Syntax

```
INT BITXOR(INT number1, INT number2)
```

Input parameters

Parameter	Data type
number1	INT
number2	INT

Function description

This function performs a bitwise XOR operation on the specified values. The input and output parameters are both of the INT type.

Examples

- Test data

a(INT)	b(INT)
2	3

- Test statements

```
SELECT BITXOR(a, b) as var1
FROM T1
```

- Test results

var1(INT)
1

5.10.2.14. CARDINALITY

This topic describes how to use the mathematical function CARDINALITY in Realtime Compute.

Syntax

```
CARDINALITY(str)
```

Input parameters

Parameter	Data type
number1	Array

Function description

This function returns the number of elements in an array.

Examples

- Test statements

```
SELECT cardinality(array[1,2,3]) AS `result`
FROM T1
```

- Test results

result(INT)
3

5.10.2.15. CONV

This topic describes how to use the built-in function CONV that converts numeral systems in Realtime Compute.

> **Note** Blink 3.2.2 and later support this function.

Syntax

```
VARCHAR CONV(BIGINT number, INT FROM_BASE, INT TO_BASE)
or
VARCHAR CONV(VARCHAR number, INT FROM_BASE, INT TO_BASE)
```

Input parameters

Parameter	Data type
number	BIGINT or VARCHAR.
FROM_BASE	INT type. This parameter cannot be negative. Valid values: **[2, 36]**.

TO_BASE	INT type. This parameter can be positive (unsigned integer), negative (signed integer), or **ABS(TO_BASE)**. Valid values: **[2, 36]**.

Description

Converts a number of the BIGINT or VARCHAR type from one numeral system to another. The return value is of the STRING type. The **CONV()** precision is 64 bits.

> **Note** If the value of the number parameter is null or an invalid character, NULL is returned.

Example

- Test data

id (INT)	x (BIGINT)	y (VARCHAR)
1	12L	'12'
2	10L	'10'
3	0L	'test'
4	NULL	NULL

- Test statements

```
SELECT  id, conv(x, 10, 16) as var1, conv(y, 10, 2) as var2
FROM T1;
```

- Test results

id (INT)	var1 (VARCHAR)	var2 (VARCHAR)
1	C	1100
2	A	1010
3	O	NULL
4	NULL	NULL

5.10.2.16. COS

This topic describes how to use the mathematical function COS in Realtime Compute.

Syntax

```
DOUBLE COS(A)
```

Input parameters

Parameter	Data type

A	DOUBLE

Function description

This function returns the cosine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.8
0.4

- Test statements

```
SELECT COS(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
0.6967067093471654
0.9210609940028851

5.10.2.17. COT

This topic describes how to use the mathematical function COT in Realtime Compute.

Syntax

```
DOUBLE COT(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the cotangent value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.8
0.4

- Test statements

```
SELECT COT(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

5.10.2.18. EXP

This topic describes how to use the mathematical function EXP in Realtime Compute.

Syntax

```
DOUBLE EXP()
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns e raised to the power of the specified number. The constant e is the base of natural logarithms.

Examples

- Test data

in1(DOUBLE)
8.0
10.0

- Test statements

```
SELECT EXP(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
2980.9579870417283

```
22026.465794806718
```

5.10.2.19. E

This topic describes how to use the mathematical function E in Realtime Compute.

Syntax

```
DOUBLE E(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the DOUBLE type value of natural constant e.

Examples

- Test data

in1(DOUBLE)
8.0
10.0

- Test statements

```
SELECT id, e() as dou1, E() as dou2
FROM T1
```

- Test results

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	2.718281828459045	2.718281828459045

5.10.2.20. FLOOR

This topic describes how to use the mathematical function FLOOR in Realtime Compute.

Syntax

```
B FLOOR(A)
```

Input parameters

Parameter	Data type
A	INT, BIGINT, FLOAT, or DOUBLE

Function description

This function rounds down the decimal portion of input parameter A and returns the largest integer less than or equal to input parameter A. The data type of output parameter B is the same as that of input parameter A.

Examples

- Test data

in1(DOUBLE)	in2(BIGINT)
8.123	3

- Test statements

```
SELECT
FLOOR(in1) as out1,
FLOOR(in2) as out2
FROM T1
```

- Test results

out1(DOUBLE)	out2(BIGINT)
8.0	3

5.10.2.21. LN

This topic describes how to use the mathematical function LN in Realtime Compute.

Syntax

```
DOUBLE ln(DOUBLE number)
```

Input parameters

Parameter	Data type
number	DOUBLE **Note** If the input parameter is of the VARCHAR or BIGINT type, it is implicitly converted to the DOUBLE type for computation.

Function description

This function returns the natural logarithm of the specified number. The return value is a logarithm of the DOUBLE type.

Examples

- Test data

ID(INT)	X(DOUBLE)
1	100.0
2	8.0

- Test statements

```
SELECT id, ln(x) as dou1, ln(e()) as dou2
FROM T1
```

- Test results

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	4.605170185988092	1.0
2	2.0794415416798357	1.0

5.10.2.22. LOG

This topic describes how to use the mathematical function LOG in Realtime Compute.

Syntax

```
DOUBLE LOG(DOUBLE base, DOUBLE x)
DOUBLE LOG(DOUBLE x)
```

Input parameters

Parameter	Data type
base	DOUBLE
x	DOUBLE

Function description

This function returns the logarithm of x to the specified base. The return value is a logarithm of the DOUBLE type. If base is not specified, this function returns the logarithm of x to base e.

Examples

- Test data

ID(INT)	BASE(DOUBLE)	X(DOUBLE)
1	10.0	100.0
2	2.0	8.0

- Test statements

```
SELECT id, LOG(base, x) as dou1, LOG(2) as dou2
FROM T1
```

- Test results

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	2.0	0.6931471805599453
2	3.0	0.6931471805599453

5.10.2.23. LOG10

This topic describes how to use the mathematical function LOG10 in Realtime Compute.

Syntax

```
DOUBLE LOG10(DOUBLE x)
```

Input parameters

Parameter	Data type
x	DOUBLE

Function description

This function returns the base-10 logarithm of x. If x is NULL, the return value is NULL. If x is negative, an exception occurs.

Examples

- Test data

id(INT)	X(INT)
1	100
2	10

- Test statements

```
SELECT id, log10(x) as dou1
FROM T1
```

- Test results

id(INT)	dou1(DOUBLE)
1	2.0
2	1.0

5.10.2.24. LOG2

This topic describes how to use the mathematical function LOG2 in Realtime Compute.

Syntax

```
DOUBLE LOG2(DOUBLE x)
```

Input parameters

Parameter	Data type
x	DOUBLE

Function description

This function returns the base-2 logarithm of x. If x is NULL, the return value is NULL. If x is negative, an exception occurs.

Examples

- Test data

id(INT)	X(INT)
1	8
2	2

- Test statements

```
SELECT id, log2(x) as dou1
FROM T1
```

- Test results

id(INT)	dou1(DOUBLE)
1	3.0
2	1.0

5.10.2.25. PI

This topic describes how to use the mathematical function PI in Realtime Compute.

Syntax

```
DOUBLE PI()
```

Function description

This function returns the value of Pi.

Examples

- Test data

ID(INT)	X(INT)
1	8

- Test statements

```
SELECT id, PI() as dou1
FROM T1
```

- Test results

ID(INT)	dou1(DOUBLE)
1	3.141592653589793

5.10.2.26. POWER

This topic describes how to use the mathematical function POWER in Realtime Compute.

Syntax

```
DOUBLE POWER(A, B)
```

Input parameters

Parameter	Data type
A	DOUBLE
B	DOUBLE

Function description

This function returns the result of A raised to the power of B. The result is a DOUBLE value.

Examples

- Test data

in1(DOUBLE)	in2(DOUBLE)
2.0	4.0

- Test statements

```
SELECT POWER(in1, in2) as aa
FROM T1
```

- Test results

aa(DOUBLE)

16.0

5.10.2.27. RAND

This topic describes how to use the mathematical function RAND in Realtime Compute.

Syntax

```
DOUBLE RAND([BIGINT seed])
```

Input parameters

Parameter	Data type
seed	BIGINT

> **Note** The value of seed is a random number, which determines the start value of a random number sequence.

Function description

This function returns a random number between 0 (inclusive) and 1 (exclusive). The return value is of the DOUBLE type.

Examples

- Test data

id(INT)	X(INT)
1	8

- Test statements

```
SELECT id, rand(1) as dou1, rand(3) as dou2
FROM T1
```

- Test results

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	0.7308781907032909	0.731057369148862

5.10.2.28. SIN

This topic describes how to use the mathematical function SIN in Realtime Compute.

Syntax

```
DOUBLE SIN(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the sine value of input parameter A.

Examples

- Test data

in1(DOUBLE)
8.0
0.4

- Test statements

```
SELECT SIN(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
0.9893582466233818
0.3894183423086505

5.10.2.29. SQRT

This topic describes how to use the mathematical function SQRT in Realtime Compute.

Syntax

```
DOUBLE SQRT(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the square root of input parameter A.

Examples

- Test data

in1(DOUBLE)
8.0

- Test statements

```
SELECT SQRT(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
2.8284271247461903

5.10.2.30. TAN

This topic describes how to use the mathematical function TAN in Realtime Compute.

Syntax

```
DOUBLE TAN(A)
```

Input parameters

Parameter	Data type
A	DOUBLE

Function description

This function returns the tangent value of input parameter A.

Examples

- Test data

in1(DOUBLE)
0.8
0.4

- Test statements

```
SELECT TAN(in1) as aa
FROM T1
```

- Test results

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

5.10.2.31. CEIL

This topic describes how to use the mathematical function CEIL in Realtime Compute.

Syntax

```
B CEIL(A)
```

Input parameters

Parameter	Data type
A	INT, BIGINT, FLOAT, or DOUBLE
B	INT, BIGINT, FLOAT, or DOUBLE

Function description

This function rounds input parameter A up to the nearest integer greater than or equal to A. The data type of output parameter B is the same as that of input parameter A.

Examples

- Test data

in1(INT)	in2(DOUBLE)
1	2.3

- Test statements

```
SELECT
CEIL(in1) as out1
CEIL(in2) as out2
FROM T1
```

- Test results

out1(INT)	out2(DOUBLE)
1	3.0

5.10.2.32. CHARACTER_LENGTH

This topic describes how to use the mathematical function CHARACTER_LENGTH in Realtime Compute.

Syntax

```
INTEGER CHARACTER_LENGTH( VARCHAR x )
```

Input parameters

Parameter	Data type
x	VARCHAR

Function description

This function returns the number of characters contained in string x.

Examples

- Test data

ID(INT)	X(VARCHAR)
1	StreamCompute

- Test statements

```
SELECT CHARACTER_LENGTH( x ) as result
FROM T1
```

- Test results

ID(INT)	result(INT)
1	13

5.10.2.33. DEGREES

This topic describes how to use the mathematical function DEGREES in Realtime Compute.

Syntax

```
DOUBLE DEGREES( double x )
```

Input parameters

Parameter	Data type
x	DOUBLE

Function description

This function converts a radian value x to a degree value.

Examples

- Test statements

```
SELECT DEGREES( PI() ) as result
FROM T1
```

- Test results

result(DOUBLE)

180.0

5.10.2.34. MOD

This topic describes how to use the mathematical function MOD in Realtime Compute.

Syntax

```
INTEGER MOD(INTEGER x,INTEGER y)
```

Input parameters

Parameter	Data type
x	INTEGER
y	INTEGER

Function description

This function returns the remainder of integer x divided by integer y. When x is negative , the result is negative.

Examples

- Test data

X(INT)	Y(INT)
29	3
-29	3
-29	-3

- Test statements

```
SELECT MOD(x,y) as result
FROM T1
```

- Test results

result(INT)
2
-2
-2

5.10.2.35. ROUND

This topic describes how to use the mathematical function ROUND in Realtime Compute for Apache Flink.

Syntax

```
T ROUND( T x, INT n)
```

Input parameters

Parameter	Data type
x	The data type of the T parameter. Valid values: DECIMAL, TINYINT, SMALLINT, INT, BIGINT, FLOAT, and DOUBLE.
n	INT.

Description

Rounds the x parameter to n decimal places.

Example 1

- Test data Table 1. T1

in1(DECIMAL)
0.7173560908995228
0.4

- Test statements

```
SELECT ROUND(in1,2) as `result`
FROM T1;
```

- Test results

result(DECIMAL)
0.72
0.40

Example 2

- Test data Table 2. T2

in2(DOUBLE)
0.7173560908995228
0.4

- Test statements

```
SELECT ROUND(in2,2) as `result`
FROM T2;
```

- Test results

result(DOUBLE)
0.72
0.4

5.10.3. Date functions

5.10.3.1. LOCALTIMESTAMP

This topic describes how to use the string function LOCALTIMESTAMP in Realtime Compute.

Syntax

```
timestamp LOCALTIMESTAMP
```

Input parameters

- None

Function description

This function returns the current timestamp of the system.

Examples

- Test statements

```
SELECT
LOCALTIMESTAMP as `result`
FROM T1
```

- Test results

result (TIMESTAMP)
2018-07-27 14:04:38.998

5.10.3.2. CURRENT_DATE

This topic describes how to use the date function CURRENT_DATE in Realtime Compute.

Syntax

```
CURRENT_DATE
```

Function description

This function returns the current system date.

Examples

- Test statements

```
SELECT CURRENT_DATE as res
FROM T1
```

- Test results

res(DATE)
2018-09-20

5.10.3.3. CURRENT_TIMESTAMP

This topic describes how to use the date function CURRENT_TIMESTAMP in Realtime Compute for Apache Flink.

Syntax

```
TIMESTAMP CURRENT_TIMESTAMP
```

> **Note** In Blink versions earlier than 3.6.0, the syntax is TIMESTAMP CURRENT_TIMESTAMP ().

Description

Returns the current UTC timestamp in milliseconds.

Example

- Test statements

```
SELECT CURRENT_TIMESTAMP as var1
FROM T1;
```

- Test results

var1(TIMESTAMP)
2007-04-30 13:10:02.047

5.10.3.4. DATEDIFF

This topic describes how to use the date function DATEDIFF in Realtime Compute.

> **Note** We recommend that you use this function in Realtime Compute V3.3.0 and later. If your Realtime Compute is earlier than V3.3.0, the return value of this function may be inaccurate.

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)
```

Input parameters

Parameter	Data type
startdate	TIMESTAMP or VARCHAR
enddate	TIMESTAMP or VARCHAR

Note The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.

Description

Calculates the number of days between the end date and start date. The return value is an integer. If the input parameter is null or a parsing error occurs, the return value is null.

Example

- Test data

datetime1 (VARCHAR)	datetime2 (VARCHAR)	nullstr (VARCHAR)
2017-10-15 00:00:00	2017-09-15 00:00:00	null

- Test statements

```
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 in
t7
FROM T1;
```

- Test results

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)	int7 (INT)
30	31	-31	null	null	null	31

5.10.3.5. DATE_ADD

This topic describes how to use the date function DATE_ADD in Realtime Compute.

Syntax

```
VARCHAR DATE_ADD(VARCHAR startdate, INT days)
VARCHAR DATE_ADD(TIMESTAMP time, INT days)
```

Input parameters

Parameter	Data type

startdate	TIMESTAMP or VARCHAR **Note** The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
enddate	TIMESTAMP
days	INT

Function description

This function adds an interval (specified by days) to the specified date and returns a new date. The return value is a VARCHAR type date in `yyyy-MM-dd` format. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

datetime1(VATCHAR)	nullstr(VATCHAR)
2017-09-15 00:00:00	null

- Test statements

```
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
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-10-15	2017-10-15	null

5.10.3.6. DATE_FORMAT

This topic describes how to use the date function DATE_FORMAT in Realtime Compute for Apache Flink.

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

Parameter	Data type

date	VARCHAR **Note** The default date format is **yyyy-MM-dd HH:mm:ss**.
time	TIMESTAMP
from_format	VARCHAR
to_format	VARCHAR

Description

Converts a VARCHAR type date from the source format to the required format. The time or date parameter specifies the source string. The **from_format** parameter is optional. It specifies the format of the source string. The default format is **yyyy-MM-dd hh:mm:ss**. The to_format parameter specifies the format of the date that you want to return. The return value is a VARCHAR type date in the required format. If any input parameter is null or a parsing error occurs, the return value is null.

Example

- Test data

date1(VARCHAR)	datetime1(VARCHAR)	nullstr(VARCHAR)
0915-2017	2017-09-15 00:00:00	null

- Test statements

```
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;
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)
170915	null	null	null	20170915	170915

5.10.3.7. DATE_SUB

This topic describes how to use the date function DATE_SUB in Realtime Compute.

Syntax

```
VARCHAR DATE_SUB(VARCHAR startdate, INT days)
VARCHAR DATE_SUB(TIMESTAMP time, INT days)
```

Input parameters

Parameter	Data type
startdate	VARCHAR **Note** The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
time	TIMESTAMP
days	INT

Function description

This function subtracts an interval (specified by days) from the specified date and returns a new date. The return value is a VARCHAR type date in yyyy-MM-dd format. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

date1(VARCHAR)	nullstr(VARCHAR)
2017-10-15	null

- Test statements

```
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
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15	2017-09-15	null

5.10.3.8. DAYOFMONTH

This topic describes how to use the date function DAYOFMONTH in Realtime Compute.

Syntax

```
BIGINT DAYOFMONTH(TIMESTAMP time)
BIGINT DAYOFMONTH(DATE date)
```

Input parameters

Parameter	Data type
date	DATE
time	TIMESTAMP

Function description

This function returns the day in the specified date or time value. The return value ranges from 1 to 31.

Examples

- Test data

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

- Test statements

```
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
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
15	22	10	15	15	15

5.10.3.9. EXTRACT

This topic describes how to use the date function EXTRACT in Realtime Compute.

Syntax

```
BIGINT EXTRACT(unit FROM time)
```

Input parameters

Parameter	Data type
time	Any date expression.

unit	Valid values of unit are as follows: • MICROSECOND • SECOND • MINUTE • HOUR • DAY • WEEK • MONTH • QUARTER • YEAR • SECOND_MICROSECOND • MINUTE_MICROSECOND • MINUTE_SECOND • HOUR_MICROSECOND • HOUR_SECOND • HOUR_MINUTE • DAY_MICROSECOND • DAY_SECOND • DAY_MINUTE • DAY_HOUR • YEAR_MONTH

Function description

This function returns one or two separate parts from the date or time value, for example, the year, month, day, hour, minute, or week.

Examples

- Test statements

```
EXTRACT(YEAR FROM CURRENT_TIMESTAMP) AS OrderYear,
EXTRACT(MONTH FROM CURRENT_TIMESTAMP) AS OrderMonth,
EXTRACT(DAY FROM CURRENT_TIMESTAMP) AS OrderDay,
EXTRACT(WEEK FROM CURRENT_TIMESTAMP) AS OrderWeek
```

- Test results

OrderYear(BIGINT)	OrderMonth(BIGINT)	OrderDay(BIGINT)	OrderWeek(BIGINT)
2018	10	11	41

5.10.3.10. FROM_UNIXTIME

This topic describes how to use the date function FROM_UNIXTIME in Realtime Compute.

Syntax

```
VARCHAR FROM_UNIXTIME(BIGINT unixtime[, VARCHAR format])
```

Input parameters

Parameter	Data type
unixtime	BIGINT
format	VARCHAR

Function description

- This function converts the timestamp (in seconds) specified by unixtime to a VARCHAR type date in the specified date format. The default format is yyyy-MM-dd HH:mm:ss.
- If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

unixtime1(INT)	nullstr(VARCHAR)
1505404800	null

- Test statements

```
SELECT FROM_UNIXTIME(unixtime1) as var1,
 FROM_UNIXTIME(unixtime1,'MMdd-yyyy') as var2,
 FROM_UNIXTIME(unixtime1,nullstr) as var3
FROM T1
```

- Test results

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-10-15	2017-10-15	null

5.10.3.11. HOUR

This topic describes how to use the date function HOUR in Realtime Compute.

Syntax

```
BIGINT HOUR(TIME time)
BIGINT HOUR(TIMESTAMP timestamp)
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Function description

This function returns the hours (in 24-hour format) in the specified time or timestamp value as a number. The return value ranges from 0 to 23.

Examples

- Test data

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

- Test statements

```
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
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
23	23	22	11	22	11

5.10.3.12. LOCALTIME

This topic describes how to use the date function LOCALTIME in Realtime Compute.

Syntax

```
TIME LOCALTIME
```

Function description

This function returns the current time of the TIME type in the session time zone. You can use LOCALTIME as a variable.

Examples

- Test statements

```
SELECT LOCALTIME as `result`
FROM T1
```

- Test results

result(TIME)
19:00:47

5.10.3.13. MINUTE

This topic describes how to use the date function MINUTE in Realtime Compute.

Syntax

```
BIGINT MINUTE(TIME time)
BIGINT MINUTE(TIMESTAMP timestamp)
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Function description

This function returns the minutes in the specified time or timestamp value as a number. The return value ranges from 0 to 59.

Examples

- Test data

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

- Test statements

```
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
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	30	23	12	23	12

5.10.3.14. MONTH

This topic describes how to use the date function MONTH in Realtime Compute for Apache Flink.

Syntax

```
BIGINT MONTH(TIMESTAMP timestamp)
BIGINT MONTH(DATE date)
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Description

Returns the month component of a specified time value. The return value ranges from 1 to 12.

Example

- Test data

a(TIMESTAMP)	b(DATE)
2016-09-15 00:00:00	2017-10-15

- Test statements

```
SELECT
 MONTH(cast( a as TIMESTAMP)) as int1,
 MONTH(cast( b as DATE)) as int2
FROM T1;
```

- Test results

int1(BIGINT)	int2(BIGINT)
9	10

5.10.3.15. NOW

This topic describes how to use the date function NOW in Realtime Compute.

Syntax

```
BIGINT NOW()
```

Input parameters

If no input parameter is specified, the UNIX timestamp (in seconds) of the current system time is returned.

Function description

This function returns the UNIX timestamp (in seconds) in the current time zone. You can specify an INT type parameter as an offset (in seconds) and add the offset to the current timestamp to return a value. For example, the NOW(100) function adds 100 seconds to the current timestamp and returns a value of the BIGINT type.

Examples

- Test data

b(VARCHAR)
null

- Test statements

```
SELECT
 NOW() as big1,
 NOW(b) as big2
FROM T1
```

- Test results

big1(BIGINT)	big2(BIGINT)
1403006911	null

5.10.3.16. SECOND

This topic describes how to use the date function SECOND in Realtime Compute.

Syntax

```
BIGINT SECOND(TIMESTAMP timestamp)
 BIGINT SECOND(TIME time)
```

Input parameters

Parameter	Data type
time	TIME
timestamp	TIMESTAMP

Function description

This function returns the seconds in the specified time value as a number. The return value ranges from 0 to 59.

Examples

- Test data

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

- Test statements

```
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
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	33	24	13	24	13

5.10.3.17. TIMESTAMPADD

This topic describes how to use the date function TIMESTAMPADD in Realtime Compute.

Syntax

```
TIMESTAMP TIMESTAMPADD(interval,INT int_expr,TIMESTAMP datetime_expr)
DATE TIMESTAMPADD(interval,INT int_expr,DATE datetime_expr)
```

Input parameters

Parameter	Data type
interval	VARCHAR
int_expr	INT
datetime_expr	TIMESTAMP or DATE

Note

The following table lists the valid units of interval.

Unit of interval	Description
FRAC_SECOND	Millisecond
SECOND	Second
MINUTE	Minute
HOUR	Hour
DAY	Day
WEEK	Week
MONTH	Month
QUARTER	Quarter
YEAR	Year

Function description

This function adds the integer expression int_expr to the date or datetime expression datetime_expr, and returns the current time of the TIME type in the session time zone. The data type of the return value of this function is the same as that of datetime_expr.

Examples

- Test data

a(TIMESTAMP)	b(DATE)
2018-07-09 10:23:56	1990-02-20

- Test statements

```
SELECT
TIMESTAMPADD(HOUR,3,a) AS `result1`
TIMESTAMPADD(DAY,3,b) AS `result2`
FROM T1
```

- Test results

result1(TIMESTAMP)	result2(DATE)
2018-07-09 13:23:56.0	1990-02-23

5.10.3.18. TO_DATE

This topic describes how to use the date function TO_DATE in Realtime Compute.

Syntax

```
Date TO_DATE(INT time)
Date TO_DATE(VARCHAR date)
Date TO_DATE(VARCHAR date, VARCHAR format)
```

Input parameters

Parameter	Data type
time	INT **Note** This parameter specifies the number of days that have elapsed since 00:00:00 Thursday, 1 January, 1970.
date	VARCHAR **Note** The default format is yyyy-MM-dd.
format	VARCHAR

Function description

This function converts a date of the INT or VARCHAR type to a date of the DATE type.

Examples

- Test data

date1(INT)	date2(VARCHAR)	date3(VARCHAR)
100	2017-09-15	20170915

- Test statements

```
SELECT TO_DATE(date1) as var1,
 TO_DATE(date2) as var2,
 TO_DATE(date3,'yyyy-MM-dd') as var3
FROM T1
```

- Test results

var1(DATE)	var2(DATE)	var3(DATE)
1970-04-11	2017-09-15	2017-09-15

5.10.3.19. TO_TIMESTAMP

This topic describes how to use the date function TO_TIMESTAMP in Realtime Compute for Apache Flink.

Syntax

```
TIMESTAMP TO_TIMESTAMP(BIGINT time)
TIMESTAMP TO_TIMESTAMP(VARCHAR date)
TIMESTAMP TO_TIMESTAMP(VARCHAR date, VARCHAR format)
```

Input parameters

Parameter	Data type
time	BIGINT **Note** The unit is milliseconds.
date	VARCHAR **Note** The default format is `yyyy-MM-dd HH:mm:ss`. If your date value is not in this format, use a Java UDF to convert the format. For more information, see UDF.
format	VARCHAR

Description

Converts the type of a date from BIGINT or VARCHAR to TIMESTAMP.

Example

- Test data

timestamp1 (BIGINT)	timestamp2 (VARCHAR)	timestamp3 (VARCHAR)
1513135677000	2017-09-15 00:00:00	20170915000000

- Test statements

```
SELECT TO_TIMESTAMP(timestamp1) as var1,
 TO_TIMESTAMP(timestamp2) as var2,
 TO_TIMESTAMP(timestamp3, 'yyyyMMddHHmmss') as var3
FROM T1;
```

- Test results

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

5.10.3.20. UNIX_TIMESTAMP

This topic describes how to use the date function UNIX_TIMESTAMP in Realtime Compute.

Syntax

```
BIGINT UNIX_TIMESTAMP()
BIGINT UNIX_TIMESTAMP(VARCHAR date)
BIGINT UNIX_TIMESTAMP(TIMESTAMP timestamp)
BIGINT UNIX_TIMESTAMP(VARCHAR date, VARCHAR format)
```

Input parameters

Parameter	Data type
timestamp	TIMESTAMP
date	VARCHAR **Note** The default date format is `yyyy-MM-dd HH:mm:ss`.
format	VARCHAR **Note** The default format is `yyyy-MM-dd hh:mm:ss`.

Function description

This function converts the specified date to a UNIX timestamp (in seconds) of the BIGINT type. If no input parameter is specified, the UNIX timestamp (in seconds) of the current time is returned. In this case, this function has the same semantics as NOW. If any input parameter is NULL or a parsing error occurs, the return value is NULL.

Examples

- Test data

nullstr(VARCHAR)
null

- Test statements

```
SELECT UNIX_TIMESTAMP() as big1,
       UNIX_TIMESTAMP(nullstr) as big2
FROM T1
```

- Test results

big1(BIGINT)	big2(BIGINT)
1403006911	null

5.10.3.21. WEEK

This topic describes how to use the date function WEEK in Realtime Compute.

Syntax

```
BIGINT WEEK(DATE date)
BIGINT WEEK(TIMESTAMP timestamp)
```

Input parameters

Parameter	Data type
date	DATE
timestamp	TIMESTAMP

Function description

This function computes the week number of the specified date in a year. The week number ranges from 1 to 53.

Examples

- Test data

dateStr(VARCHAR)	date1(DATE)	ts1(TIMESTAMP)
2017-09-15	2017-11-10	2017-10-15 00:00:00

- Test statements

```
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
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)
37	45	41	37

5.10.3.22. YEAR

This topic describes how to use the date function YEAR in Realtime Compute.

Syntax

```
BIGINT YEAR(TIMESTAMP timestamp)
BIGINT YEAR(DATE date)
```

Input parameters

Parameter	Data type
date	DATE

timestamp	TIMESTAMP

Function description

This function returns the year in the specified time value.

Examples

- Test data

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

- Test statements

```
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
```

- Test results

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
2016	2017	2017	2017	2015	2017

5.10.4. Logical functions

5.10.4.1. =

This topic describes how to use the logical operation function = of Realtime Compute.

Syntax

```
A = B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 = int2;
```

- Test result

aa (INT)
97

5.10.4.2. >

This topic describes how to use the logical operation function > of Realtime Compute.

Syntax

```
A > B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is greater than B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	65	100

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 > int2;
```

- Test result

aa (INT)
97

5.10.4.3. >=

This topic describes how to use the logical operation function >= of Realtime Compute.

Syntax

```
A >= B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is greater than or equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65
9	6	61

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 >= int2;
```

- Test result

aa (INT)
97
9

5.10.4.4. <=

This topic describes how to use the logical operation function <= of Realtime Compute.

Syntax

```
A <= B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is smaller than or equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 <= int2;
```

- Test result

aa (INT)
97
9

5.10.4.5. <

This topic describes how to use the logical operation function < of Realtime Compute.

Syntax

```
A < B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is smaller than B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 < int2;
```

- Test result

aa (INT)
97
9

5.10.4.6. <>

This topic describes how to use the logical operation function <> of Realtime Compute.

Syntax

```
A <> B
```

Input parameter

Name	Data type
A	INT
B	INT

Function description

TRUE is returned if A is not equal to B. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
97	66	6

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int3 <> int2;
```

- Test result

aa (INT)
97

5.10.4.7. AND

This topic describes how to use the logical operation function AND of Realtime Compute.

Syntax

```
A AND B
```

Input parameter

Name	Data type
A	BOOLEAN
B	BOOLEAN

Function description

TRUE is returned if both A and B are TRUE. Otherwise, FALSE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1=255 AND int3=65;
```

- Test result

aa (INT)
97

5.10.4.8. BETWEEN AND

This topic describes how to use the logical function BETWEEN AND in Realtime Compute for Apache Flink.

Syntax

```
A BETWEEN B AND C
```

Input parameters

Parameter	Data type
A	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
B	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
C	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME

Description

Selects a value within a data range defined by another two values.

Example 1

- Test data

int1(INT)	int2(INT)	int3(INT)
90	80	100
11	10	7

- Test statements

```
SELECT int1 as aa
FROM T1
WHERE int1 BETWEEN int2 AND int3;
```

- Test results

aa(int)
90

Example 2

- Test data

var1(varchar)	var2(varchar)	var3(varchar)
b	a	c

- Test statements

```
SELECT var1 as aa
FROM T1
WHERE var1 BETWEEN var2 AND var3;
```

- Test results

aa(varchar)
b

Example 3

- Test data

TIMESTAMP1(TIMESTAMP)	TIMESTAMP2(TIMESTAMP)	TIMESTAMP3(TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:20	1969-07-20 20:17:45

- Test statements

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- Test results

aa(TIMESTAMP)
1969-07-20 20:17:30

5.10.4.9. IS NOT FALSE

This topic describes how to use the logical operation function IS NOT FALSE of Realtime Compute.

Syntax

```
A IS NOT FALSE
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

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

Example

- Test data

int1 (INT)	int2 (INT)

255	97

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS NOT FALSE;
```

- Test result

aa (INT)
97

5.10.4.10. IS NOT NULL

This topic describes how to use the logical function IS NOT NULL in Realtime Compute.

Syntax

```
value IS NOT NULL
```

Input parameters

Parameter	Data type
value	Any data type

Description

If the value is `null`, `false` is returned. Otherwise, `true` is returned.

Example

- Test data

int1 (INT)	int2 (VARCHAR)
97	null
9	ww123

- Test statements

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NOT NULL;
```

- Test results

aa (INT)
9

5.10.4.11. IS NOT TRUE

This topic describes how to use the logical operation function IS NOT TRUE of Realtime Compute.

Syntax

```
A IS NOT TRUE
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

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

Example

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int1=25 IS NOT TRUE;
```

- Test result

aa (INT)
97

5.10.4.12. IS NOT UNKNOWN

This topic describes how to use the logical operation function IS NOT UNKNOWN of Realtime Compute.

Syntax

```
A IS NOT UNKNOWN
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

A is a logical comparison expression, such as 6 < 8.

In normal cases, when A compares two numbers, the value of A can be determined, which is either TRUE or FALSE. However, if either operand is not a number, the value of A cannot be determined. `IS NOT UNKNOWN` is used to determine whether this situation occurs. If the value of A cannot be determined (that is, the value is neither `TRUE` nor `FALSE`), `FALSE` is returned. If the value of A can be determined (that is, the value is `TRUE` or `FALSE`), `TRUE` is returned.

Example 1

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS NOT UNKNOWN;
```

- Test result

aa (INT)
97

Example 2

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1 < null IS NOT UNKNOWN;
```

- Test result

aa (INT)
null

5.10.4.13. IS NULL

This topic describes how to use the logical operation function IS NULL of Realtime Compute.

Syntax

```
value IS NULL
```

Input parameter

Name	Data type
value	Any data type

Function description

If the value is `NULL`, `TRUE` is returned. Otherwise, `FALSE` is returned.

Example

- Test data

int1 (INT)	int2 (VARCHAR)
97	NULL
9	www

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NULL;
```

- Test result

aa (INT)
97

5.10.4.14. IS TRUE

This topic describes how to use the logical function IS TRUE in Realtime Compute for Apache Flink.

Syntax

```
A IS TRUE
```

Input parameters

Parameter	Data type
A	BOOLEAN

Description

If A is true, true is returned. If A is false, false is returned.

Example

- Test data

int1(INT)	int2(INT)
255	97

- Test statements

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS TRUE;
```

- Test results

aa(int)
97

5.10.4.15. IS UNKNOWN

This topic describes how to use the logical operation function IS UNKNOWN of Realtime Compute.

Syntax

```
A IS UNKNOWN
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

If the value of A (a logical comparison expression) cannot be determined (that is, the value is neither `TRUE` nor `FALSE`), `TRUE` is returned. If the value of A can be determined (that is, the value is `TRUE` or `FALSE`), `FALSE` is returned. In normal cases, when A compares two numbers (for example, `6<>8`), the value of A can be determined, which is either TRUE or FALSE. However, if either operand is not a number, the value of A cannot be determined. `IS UNKNOWN` is used to determine whether this situation occurs.

Example 1

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS UNKNOWN;
```

- Test result

aa (INT)
null

Example 2

- Test data

int1 (INT)	int2 (INT)
255	97

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1 > null IS UNKNOWN;
```

- Test result

aa (INT)
97

5.10.4.16. LIKE

This topic describes how to use the logical operation function LIKE of Realtime Compute.

Syntax

```
A LIKE B
```

Input parameter

Name	Data type
A	VARCHAR
B	VARCHAR

Function description

TRUE is returned if A matches B. Otherwise, FALSE is returned.

> **Note** You can use the percent sign (%) as a wildcard.

Example 1

- Test data

int1 (INT)	VARCHAR2 (VARCHAR)	VARCHAR3 (VARCHAR)

90	ss97	97ss
99	ss10	7ho7

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR2 LIKE 'ss%';
```

- Test result

aa (INT)
90
99

Example 2

- Test data

int1 (INT)	VARCHAR2 (VARCHAR)	VARCHAR3 (VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ho%';
```

- Test result

aa (INT)
99

5.10.4.17. NOT

This topic describes how to use the logical operation function NOT of Realtime Compute.

Syntax

```
NOT A
```

Input parameter

Name	Data type
A	BOOLEAN

Function description

If A is `TRUE`, `FALSE` is returned. If A is `FALSE`, `TRUE` is returned.

Example

- Test data

int2 (INT)	int3 (INT)
97	65

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE NOT int3=62;
```

- Test result

aa (INT)
97

5.10.4.18. NOT BETWEEN AND

This topic describes how to use the logical operation function NOT BETWEEN AND of Realtime Compute.

Syntax

```
A NOT BETWEEN B AND C
```

Input parameter

Name	Data type
A	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
B	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME
C	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME

Function description

This function selects a value not within a data range defined by two other values.

Example 1

- Test data

int1 (INT)	int2 (INT)	int3 (INT)

90	97	80
11	10	7

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE int1 NOT BETWEEN int2 AND int3;
```

- Test result

aa (INT)
11

Example 2

- Test data

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
d	a	c

- Test statement

```
SELECT int1 as aa
FROM T1
WHERE var1 NOT BETWEEN var2 AND var3;
```

- Test result

aa (VARCHAR)
d

Example 3

- Test data

TIMESTAMP1 (TIMESTAMP)	TIMESTAMP2 (TIMESTAMP)	TIMESTAMP3 (TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:40	1969-07-20 20:17:45

- Test statement

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 NOT BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- Test result

aa (TIMESTAMP)
1969-07-20 20:17:30

5.10.4.19. IN

This topic describes how to use the logical operation function IN of Realtime Compute.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

Input parameter

Name	Data type
value1	Constant
value2	Constant

Function description

This function queries records that match the input parameters.

Example

- Test data

id (INT)	LastName (VARCHAR)
1	Adams
2	Bush
3	Carter

- Test statement

```
SELECT *
FROM T1
WHERE LastName IN ('Adams','Carter')
```

- Test result

id (INT)	LastName (VARCHAR)
1	Adams
3	Carter

5.10.4.20. OR

This topic describes how to use the logical operation function OR of Realtime Compute.

Syntax

```
A OR B
```

Input parameter

Name	Data type
A	BOOLEAN
B	BOOLEAN

Function description

FALSE is returned if both A and B are FALSE. Otherwise, TRUE is returned.

Example

- Test data

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

- Test statement

```
SELECT int2 as aa
FROM T1
WHERE int1=255 OR int3=65;
```

- Test result

aa (INT)
97

5.10.4.21. IS DISTINCT FROM

This topic describes how to use the logical function IS DISTINCT FROM in Realtime Compute for Apache Flink.

Syntax

```
A IS DISTINCT FROM B
```

Input parameters

Parameter	Data type
A	Any data type
B	Any data type

Description

- If the data types or values of A and B are different, `true` is returned.

- If the data types and values of A and B are the same, `false` is returned.
- If both A and B are null, false is returned even when their data types are different.

Example

- Test data

A(int)	B(varchar)
97	97
null	sss
null	null

- Test statements

```
SELECT
A IS DISTINCT FROM B as 'result'
FROM T1;
```

- Test results

result(BOOLEAN)
true
true
false

5.10.4.22. IS NOT DISTINCT FROM

This topic describes how to use the logical operation function IS NOT DISTINCT FROM of Realtime Compute.

Syntax

```
A IS NOT DISTINCT FROM B
```

Input parameter

Name	Data type
A	Any data type
B	Any data type

Function description

- `FALSE` is returned if the data types or values of A and B are different.
- `TRUE` is returned if the data types and values of A and B are the same.
- If both A and B are null, TRUE is returned even when their data types are different.

Example

- Test data

A (INT)	B (VARCHAR)
97	97
null	sss
null	null

- Test statement

```
SELECT
A IS NOT DISTINCT FROM B as `result`
FROM T1
```

- Test result

result (BOOLEAN)
FALSE
FALSE
TRUE

5.10.4.23. NOT IN

This topic describes how to use the logical operation function NOT IN of Realtime Compute.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (value1,value2,...)
```

Input parameter

Name	Data type
value1	Constant
value2	Constant

Function description

This function queries records that do not match the input parameters.

Example

- Test data

id (INT)	LastName (VARCHAR)
1	Adams

2	Bush
3	Carter

- Test statement

```
SELECT *
FROM T1
WHERE LastName NOT IN ('Adams','Carter')
```

- Test result

id (INT)	LastName (VARCHAR)
2	Bush

5.10.5. Conditional functions

5.10.5.1. CASE WHEN

This topic describes how to use the conditional function CASE WHEN in Realtime Compute for Apache Flink.

Syntax

```
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
```

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.

Precautions

If the CASE WHEN function returns a constant string, spaces are added after the string. In the following example, if the else condition is met, the return value is `ios` followed by several spaces.

```
case when device_type = 'android'
then 'android'
else 'ios'
end as os
```

You can resolve this issue in the following two ways:

- Use the TRIM function to remove spaces. In this example, use the `trim(os)` field to replace all the `os` fields.
- Use the CAST function to convert a constant string to a string of the VARCHAR type.

Example

- Test data

device_type(VARCHAR)

android
ios
win

- Test statements
  - Use the TRIM function

```
SELECT
   trim(os), -- The TRIM function is used here.
   CHAR_LENGTH(trim(os)) -- The TRIM function is used here.
from(
   SELECT
     case when device_type = 'android'
     then 'android'
     else 'ios'
end as os
FROM T1
);
```

  - Use the CAST function

```
SELECT
 os,
CHAR_LENGTH(os)
from
(SELECT
 case when device_type = 'android'
 then cast('android' as varchar) -- The CAST function is used here.
 else cast('ios' as varchar) -- The CAST function is used here.
end as os
FROM T1
);
```

- Test results

os(VARCHAR)	length(INT)
android	7
ios	3
ios	3

5.10.5.2. COALESCE

This topic describes how to use the conditional function COALESCE in Realtime Compute.

Syntax

```
COALESCE(A,B,...)
```

Input parameters

Parameter	Data type
A	Any data type
B	Any data type

> **Note** All values must be of the same type or be NULL. Otherwise, an exception occurs.

Function description

This function returns the first non-NULL value in the specified list. The return value is of the same type as the input parameter values. If all values in the list are NULL, the return value is NULL.

> **Note** The list must contain at least one parameter. Otherwise, an exception occurs.

Examples

- Test data

var1(VARCHAR)	var2(VARCHAR)
null	30

- Test statements

```
SELECT COALESCE(var1,var2) as aa
FROM T1
```

- Test results

aa(VARCHAR)
30

5.10.5.3. IF

This topic describes how to use the conditional function IF in Realtime Compute.

Syntax

```
T IF(BOOLEAN testCondition, T valueTrue, T valueFalseOrNull)
```

> **Note** T represents a return value of any type.

Input parameters

Parameter	Data type
testCondition	BOOLEAN
valueTrue	Any data type (The valueTrue and valueFalseOrNull parameters must be of the same type.)
valueFalseOrNull	Any data type (The valueTrue and valueFalseOrNull parameters must be of the same type.)

Function description

This function uses the BOOLEAN value of testCondition as the judgment criterion. If testCondition is true, this function returns valueTrue. If testCondition is false, this function returns valueFalseOrNull. If testCondition is NULL, it is also regarded as false and this function returns valueFalseOrNull. If any other parameter is NULL, this function works based on normal semantics. The data type of the return value is determined by T.

Examples

- Test data

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	Jack	Harry
1	2	Jack	null
1	2	null	Harry

- Test statements

```
SELECT IF(int1 < int2,str1, str2) as int1
FROM T1
```

- Test results

int1(VARCHAR)
Jack
Jack
null

5.10.5.4. IS_ALPHA

This topic describes how to use the conditional function IS_ALPHA in Realtime Compute for Apache Flink.

Syntax

```
BOOLEAN IS_ALPHA(VARCHAR str)
```

Input parameters

Parameter	Data type
str	VARCHAR

Description

Checks whether the specified string contains only letters. If yes, the return value is true. If not, the return value is false.

Example

- Test data

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- Test statements

```
SELECT IS_ALPHA(e) as boo1,IS_ALPHA(f) as boo2,IS_ALPHA(g) as boo3
FROM T1;
```

- Test results

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
false	true	false

5.10.5.5. IS_DECIMAL

This topic describes how to use the conditional function IS_DECIMAL in Realtime Compute.

Syntax

```
BOOLEAN IS_DECIMAL(VARCHAR str)
```

Input parameters

Parameter	Data type
str	VARCHAR

Function description

This function checks whether the specified string can be converted to a decimal value. If yes, the return value is true. If not, the return value is false.

Examples

- Test data

a(VARCHAR)	b(VARCHAR)	c(VARCHAR)	d(VARCHAR)	e(VARCHAR)	f(VARCHAR)	g(VARCHAR)

1	123	2	11.4445	3	asd	null

- Test statements

```
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
```

- Test results

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)	boo4(BOOLEAN)	boo5(BOOLEAN)	boo6(BOOLEAN)	boo7(BOOLEAN)
true	true	true	true	true	false	false

5.10.5.6. IS_DIGIT

This topic describes how to use the conditional function IS_DIGIT in Realtime Compute.

Syntax

```
BOOLEAN IS_DIGIT(VARCHAR str)
```

Input parameters

Parameter	Data type
str	VARCHAR

Function description

This function checks whether the specified string contains only digits. If yes, the return value is true. If not, the return value is false. The return value is of the BOOLEAN type.

Examples

- Test data

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- Test statements

```
SELECT
IS_DIGIT(e) as boo1,
IS_DIGIT(f) as boo2,
IS_DIGIT(g) as boo3
FROM T1
```

- Test results

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
true	false	false

5.10.5.7. NULLIF

This topic describes how to use the conditional function NULLIF in Realtime Compute.

Syntax

```
NULLIF(A,B)
```

Input parameters

Parameter	Data type
A	INT
B	INT

Function description

This function returns NULL if the two specified parameters have the same value, and returns the value of the first parameter if the parameters have different values.

Examples

- Test data

var1(INT)	var2(INT)
30	30

- Test statements

```
SELECT NULLIF(var1,var2) as aa
FROM T1
```

- Test results

aa(INT)
null

5.10.6. Table-valued functions

5.10.6.1. GENERATE_SERIES

This topic describes how to use the table-valued function GENERATE_SERIES in Realtime Compute.

Syntax

```
GENERATE_SERIES(INT from, INT to)
```

Input parameters

Parameter	Data type
from	The lower bound of a consecutive series of values (including the lower bound) to be generated. This parameter is of the INT type.
to	The upper bound of a consecutive series of values (excluding the upper bound) to be generated. This parameter is of the INT type.

Function description

This function generates a consecutive series of values from the lower bound to the upper bound minus one.

Examples

- Test data

s(INT)	e(INT)
1	3
-2	1

- Test statements

```
SELECT s, e, v
FROM T1, lateral table(GENERATE_SERIES(s, e))
as T(v)
```

- Test results

s(INT)	e(INT)	v(INT)
1	3	1
1	3	2
-2	1	-2
-2	1	-1

-2	1	-0

5.10.6.2. JSON_TUPLE

This topic describes how to use the table-valued function JSON_TUPLE in Realtime Compute for Apache Flink.

> **Important** Only Blink supports JSON_TUPLE. Flink does not support JSON_TUPLE.

Syntax

```
JSON_TUPLE(str, path1, path2 ..., pathN)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The JSON string.
path1 to pathN	VARCHAR	A path string, which does not start with a dollar sign (`$`).

Description

Returns the value represented by each path string from the JSON string.

Example

- Test data

d(VARCHAR)	s(VARCHAR)
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	qwe3
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	qwe2

- Test statements

```
SELECT d, v
FROM T1, lateral table(JSON_TUPLE(d, 'qwe', s))
AS T(v);
```

- Test results

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

5.10.6.3. STRING_SPLIT

This topic describes how to use the table-valued function STRING_SPLIT in Realtime Compute for Apache Flink.

Syntax

```
string_split(string, separator)
```

Input parameters

Parameter	Type	Description
string	VARCHAR	The string that you want to split.
separator	VARCHAR	The specified delimiter. **Note** The delimiter must be a single string.

Description

Splits a string into rows of substrings based on a specified delimiter and returns a table that consists of rows of substrings. Before you use this function, take note of the following items:

- If the value of the string is null, this function returns an empty row.
- If the string contains two or more consecutive delimiters, this function returns a zero-length substring.
- If the string does not contain the specified delimiter, this function returns only this string.

Example

- Test data Table 1. T1

d(varchar)	s(varchar)
abc-bcd	-
hhh	-

- Test statements

```
select d,v
from T1,
lateral table(string_split(d, s)) as T(v);
```

- Test results

d(varchar)	v(varchar)
abc-bcd	abc
abc-bcd	bcd
hhh	hhh

5.10.6.4. MULTI_KEYVALUE

This topic describes how to use the table-valued function MULTI_KEYVALUE in Realtime Compute.

> **Note** This function is only available in Realtime Compute V2.2.2 and later.

Syntax

```
MULTI_KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name1, VARCHAR key_name2, ...)
```

Input parameters

Parameter	Data type	Description
str	VARCHAR	The key-value pairs in a string.
split1	VARCHAR	The separator of key-value pairs. If split1 is null, a whitespace is used as the separator between key-value pairs. If the length of split1 is greater than 1, split1 only represents a set of separators, in which each character represents a valid separator.
split2	VARCHAR	The key-value separator. If split2 is null, a whitespace is used as the key-value separator. If the length of split2 is greater than 1, split2 only represents a set of separators, in which each character represents a valid separator.
key_name1, key_name2, ...	VARCHAR	The list of keys whose values you want to obtain.

Description

Parses the key-value pairs in a string based on the key-value pair separator and key-value separator, and then returns a list of values for the key names such as key_name1 and key_name2. If a key_name does not exist, the return value is null.

Example

- Test data

str (VARCHAR)	split1 (VARCHAR)	split2 (VARCHAR)	key1 (VARCHAR)	key2 (VARCHAR)
k1=v1;k2=v2	;	=	k1	k2
null	;	=	k1	k2

k1:v1;k2:v2	;	:	k1	k3
k1:v1;k2:v2	;	=	k1	k2
k1:v1;k2:v2	,	:	k1	k2
k1:v1;k2=v2	;	:	k1	k2
k1:v1abck2:v2	cab	:	k1	k2
k1:v1;k2=v2	;	:=	k1	k2
k1:v1 k2:v2	null	:	k1	k2
k1 v1;k2 v2	;	null	k1	k2

- Test statements

```
SELECT c1, c2
FROM T1, lateral table(MULTI_KEYVALUE(str, split1, split2, key1, key2))
as T(c1, c2);
```

- Test results

c1 (VARCHAR)	c2 (VARCHAR)
v1	v2
null	null
v1	null
null	null
null	null
v1	null
v1	v2
v1	v2
v1	v2
v1	v2

5.10.7. Type conversion function

5.10.7.1. CAST

This topic describes how to use the type conversion function CAST in Realtime Compute for Apache Flink.

Syntax

```
CAST(A AS type)
```

Input parameters

Parameter	Data type
A	For more information, see Data type conversion.

Description

Converts the value of the input parameter A to a specified type. If the type after conversion is not consistent with the field type in the destination table, `Insert into: Query result and target table 'test_result' field type(s) not match.` is reported.

Example

- Test data

var1(VARCHAR)	var2(INT)
1000	30

- Test statements

```
SELECT CAST(var1 AS INT) as aa
FROM T1;
```

- Test results

aa(INT)
1000

5.10.8. Aggregate functions

5.10.8.1. AVG

This topic describes how to use the aggregate function AVG in Realtime Compute for Apache Flink. In Flink SQL, the AVG function returns the average value of all values in a specified expression.

Syntax

```
AVG(A)
```

Input parameters

Parameter	Data type
A	TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, or DOUBLE

Description

Returns the average value of all the values in a specified expression.

> **Note** The return value is of the DOUBLE type by default. If the field value in the result table is of a type other than DOUBLE, you must use CAST to convert the data type.

Example

- Test data

var1(INT)	var2(INT)
4	30
6	30

- Test statements

```
SELECT AVG(var1) as aa
FROM T1;
```

- Test results

aa(INT)
5

5.10.8.2. CONCAT_AGG

This topic describes how to use the aggregate function CONCAT_AGG in Realtime Compute. In Flink SQL, the CONCAT_AGG function concatenates the strings of all specified fields and returns a new string.

Syntax

```
CONCAT_AGG([linedelimiter,] value)
```

Input parameters

Parameter	Data type
linedelimiter (optional)	Only a string constant is currently supported.

Function description

This function concatenates the strings of all specified fields and returns a new string. The default connector is `\n`. The return value is of the VARCHAR type.

Examples

- Test data

c(VARCHAR)
Hi
Hi

Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi

- Test statements

```
SELECT
concat_agg(c) as var1,
concat_agg('-', c) as var2
FROM MyTable
GROUP BY c
```

- Test results

var1(VARCHAR)	var2(VARCHAR)
Hi\nHi\nHi\nHi\nHi\nHi\nHi\nHi\nHi	Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi

5.10.8.3. COUNT

This topic describes how to use the aggregate function COUNT in Realtime Compute for Apache Flink. In Flink SQL, the COUNT function returns the number of rows in a given column.

Syntax

```
COUNT(A)
```

Input parameters

Parameter	Data type
A	• Supported data types: TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, DOUBLE, BOOLEAN, and VARCHAR. • Unsupported data types: DATE, TIME, TIMESTAMP, and VARBINARY.

Example

- Test data

var1(VARCHAR)

1000
100
10
1

- Test statements

```
SELECT COUNT(var1) as aa
FROM T1;
```

- Test results

aa(BIGINT)
4

5.10.8.4. FIRST_VALUE

This topic describes how to use the aggregate function FIRST_VALUE in Realtime Compute. In Flink SQL, the FIRST_VALUE function returns the first non-null record of a data stream.

Syntax

```
T FIRST_VALUE(T value)
T FIRST_VALUE(T value, Long order)
```

Input parameters

Parameter	Data type
value	Any data type (The input parameters must be of the same type.)
order	INT

Function description

This function returns the first non-null record of a data stream. A record with the smallest order value is obtained as the first non-null record.

Examples

- Test data

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"

- Test statements

```
SELECT c,
 first_value(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED preceding
) as var1
from T1
```

- Test results

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

5.10.8.5. LAST_VALUE

This topic describes how to use the aggregate function LAST_VALUE in Realtime Compute for Apache Flink. In Flink SQL, the LAST_VALUE function returns the last non-null record of a data stream.

Syntax

```
T LAST_VALUE(T value)
T LAST_VALUE(T value, BIGINT order)
```

Input parameters

Parameter	Data type
value	Any data type **Note** All input parameters must be of the same data type.
order	BIGINT

Description

Returns the last non-null record of a data stream. The row of LAST_VALUE is determined based on the order parameter and the record with the largest order value is used as LAST_VALUE.

Example 1

- Test data

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"

- Test statements

```
SELECT c,
 LAST_VALUE(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- Test results

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

Example 2

- Test data

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
1L	1	"Hello"	5
2L	2	"Hello"	5
3L	3	"Hello"	6
4L	4	"Hello"	5
5L	5	"Hello"	6
6L	6	"Hello"	5
7L	7	"Hello World"	4
8L	8	"Hello World"	8
20L	20	"Hello World"	9

- Test statements

```
SELECT c,
 LAST_VALUE(b,order)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- Test results

c(VARCHAR)	var1(INT)
"Hello"	1
"Hello"	1

"Hello"	3
"Hello"	3
"Hello"	3
"Hello"	3
"Hello World"	7
"Hello World"	8
"Hello World"	20

5.10.8.6. MAX

This topic describes how to use the aggregate function MAX in Realtime Compute. In Flink SQL, the MAX function returns the maximum value among all input values.

Syntax

```
MAX(A)
```

Input parameters

Parameter	Data type
A	TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, DOUBLE, BOOLEAN, or VARCHAR **Note** The following data types are not supported: DATE, TIME, TIMESTAMP, and VARBINARY.

Function description

This function returns the maximum value among all input values.

Examples

- Test data

var1(INT)
4
8

- Test statements

```
SELECT MAX(var1) as aa
FROM T1
```

- Test results

aa(INT)
8

5.10.8.7. MIN

This topic describes how to use the aggregate function MIN in Realtime Compute for Apache Flink. In Flink SQL, the MIN function returns the minimum value of all input values.

Syntax

```
MIN(A)
```

Input parameters

Parameter	Data type
A	• Supported data types: TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, DOUBLE, BOOLEAN, and VARCHAR. • Unsupported data types: DATE, TIME, TIMESTAMP, and VARBINARY.

Description

Returns the minimum value of all input values.

Example

- Test data

var1(INT)
4
8

- Test statements

```
SELECT MIN(var1) as aa
FROM T1;
```

- Test results

aa(INT)
4

5.10.8.8. SUM

This topic describes how to use the aggregate function SUM in Realtime Compute for Apache Flink. In Flink SQL, the SUM function returns the sum of all input values.

Syntax

```
SUM(A)
```

Input parameters

Parameter	Data type
A	TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, or DOUBLE

Description

Returns the sum of all input values.

Example

- Test data

var1(INT)
4
4

- Test statements

```
SELECT sum(var1) as aa
FROM T1;
```

- Test results

aa(INT)
8

5.10.8.9. VAR_POP

This topic describes how to use the aggregate function VAR_POP in Realtime Compute for Apache Flink. In Flink SQL, the VAR_POP function returns the population variance of all input values in a specified expression.

Syntax

```
T VAR_POP(T value)
```

Input parameters

Parameter	Data type
value	Numeric type, such as BIGINT or DOUBLE

Description

Returns the population variance of all input values.

Example

- Test data

a(BIGINT)	c(VARCHAR)
2900	Hi
2500	Hi
2600	Hi
3100	Hello
11000	Hello

- Test statements

```
SELECT
VAR_POP(a) as `result`,
c
FROM MyTable
GROUP BY c;
```

- Test results

result(BIGINT)	c
28889	Hi
15602500	Hello

5.10.8.10. STDDEV_POP

This topic describes how to use the aggregate function STDDEV_POP in Realtime Compute. In Flink SQL, the STDDEV_POP function returns the population standard deviation of a set of values.

Syntax

```
T STDDEV_POP(T value)
```

Input parameters

Parameter	Data type
value	BIGINT or DOUBLE

Function description

This function returns the population standard deviation of a set of values.

Examples

- Test data

a(DOUBLE)	c(VARCHAR)

0	Hi
1	Hi
2	Hi
3	Hi
4	Hi
5	Hi
6	Hi
7	Hi
8	Hi
9	Hi

- Test statements

```
SELECT c, STDDEV_POP(a) as dou1
FROM MyTable
GROUP BY c
```

- Test results

c(VARCHAR)	dou1(DOUBLE)
Hi	2.8722813232690143

5.10.9. Other functions

5.10.9.1. UUID

This topic describes how to use the UUID function in Realtime Compute for Apache Flink. In Flink SQL, the UUID function returns a universally unique identifier.

Syntax

```
VARCHAR UUID()
```

Description

Returns a universally unique identifier.

Example

- Test statements

```
SELECT uuid() as `result`
FROM T1;
```

- Test results

result(VARCHAR)
a364e414-e68b-4e5c-9166-65b3a153e257

5.10.9.2. DISTINCT

This topic describes how to use the DISTINCT function in Realtime Compute for Apache Flink. The DISTINCT function is used in SELECT statements to remove duplicate query results.

Syntax

```
SELECT DISTINCT expressions
FROM tables
...
```

- `DISTINCT` must be placed before expressions. When you use `DISTINCT` with other functions at the same time, you must place the DISTINCT function at the beginning of a statement, for example, `concat_agg(DISTINCT ',' ,device_id)` .
- `expressions` can be one or more expressions, specific columns, or any other valid expressions such as functions.

Example

- Test statements
  The following example shows how to use DISTINCT in Flink SQL:

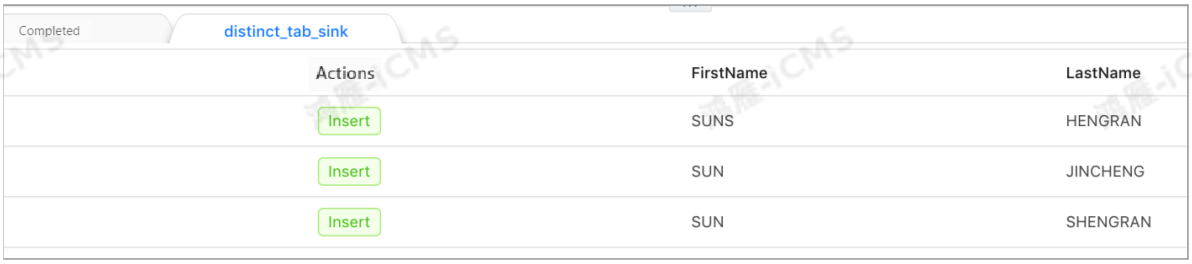
```
CREATE TABLE distinct_tab_source(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
   type='random'
) ;

CREATE TABLE distinct_tab_sink(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
    type = 'print'
) ;

INSERT INTO distinct_tab_sink
SELECT DISTINCT FirstName, LastName -- Remove duplicate records based on the FirstNam
e and LastName columns.
FROM distinct_tab_source;
```

- Test data

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN

SUN	SHENGRAN

- Test results



> **Note**
> - The test data contains four records. `DISTINCT FirstName, LastName` removes one duplicate record that contains SUNS,SHENGRAN and returns three unique records.
> - The `SUNS,HENGRAN` and `SUN,SHENGRAN` records are retained. This indicates that `DISTINCT FirstName, LastName` processes the FirstName and LastName columns separately, and does not concatenate them for deduplication.

Alternative for DISTINCT

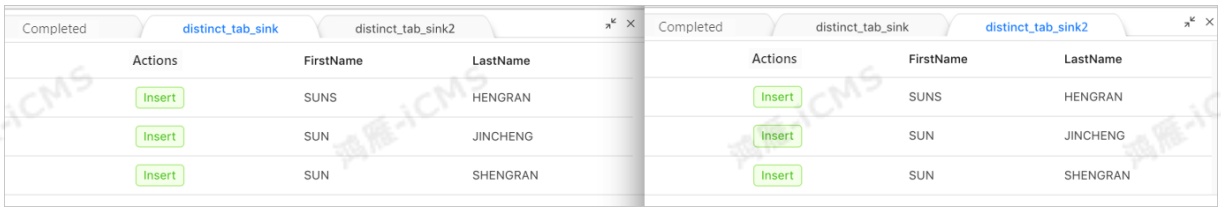
`GROUP BY` in SQL statements also provides a deduplication function similar to that of `DISTINCT`. The following example shows the syntax of `GROUP BY`:

```
SELECT expressions
FROM tables
GROUP BY expressions
;
```

The following example writes an SQL multi-insert query that has an equivalent effect to that of the DISTINCT function:

```
CREATE TABLE distinct_tab_source(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
   type='random'
);

CREATE TABLE distinct_tab_sink(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
    type = 'print'
);

CREATE TABLE distinct_tab_sink2(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
    type = 'print'
);

INSERT INTO distinct_tab_sink
    SELECT DISTINCT FirstName, LastName -- Remove duplicate records based on the FirstN
ame and LastName columns.
        FROM distinct_tab_source;

INSERT INTO distinct_tab_sink2
    SELECT FirstName, LastName
        FROM distinct_tab_source
         GROUP BY FirstName, LastName; -- Remove duplicate records based on the
FirstName and LastName columns.
```

The following figure shows that the results of GROUP BY and DISTINCT functions are the same when the same test data is used. Therefore, the semantics of GROUP BY and DISTINCT are equivalent.



Use of DISTINCT in the aggregate function COUNT

The use of `DISTINCT` enables the aggregate function `COUNT` to count the number of records after deduplication.

```
COUNT(DISTINCT expression)
```

> **Important** `expression` supports only a single expression.

Example of the COUNT DISTINCT syntax

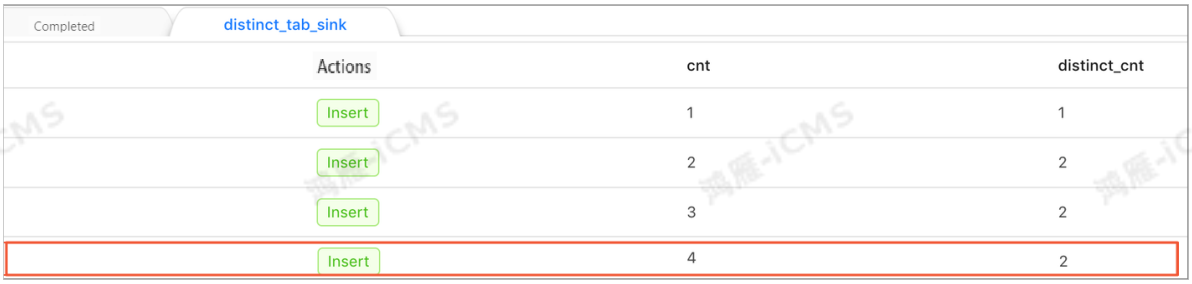
- Test statements

```
CREATE TABLE distinct_tab_source(
    FirstName  VARCHAR,
    LastName VARCHAR
)WITH(
   type='random'
) ;

CREATE TABLE distinct_tab_sink(
    cnt BIGINT  ,
    distinct_cnt BIGINT
)WITH(
    type = 'print'
) ;

INSERT INTO distinct_tab_sink
    SELECT
      COUNT(FirstName),  -- Duplicate records are not removed.
      COUNT(DISTINCT FirstName)  -- Duplicate records are removed based on the
FirstName column.
    FROM distinct_tab_source;
```

- Test data

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- Test results



5.11. UDXs

5.11.1. Overview

This topic describes how to build a development environment and use user-defined extensions (UDXs) in Realtime Compute for Apache Flink.

> **Important**
>
> - Only Realtime Compute for Apache Flink in exclusive mode supports UDXs.
> - Blink is developed based on Apache Flink SQL by Alibaba Cloud Realtime Compute for Apache Flink to improve computing performance. UDXs can be used only in Blink.

UDX type

Realtime Compute for Apache Flink supports three types of UDXs, as described in the following table.

UDX type	Description
UDF	User-defined scalar function. The relationship between the input and output of UDFs is one-to-one mapping, which indicates that one value is returned each time a UDF reads one row of data.
UDAF	User-defined aggregation function. The relationship between the input and output of UDAFs is many-to-one mapping. A UDAF aggregates multiple input records into one output record. A UDAF can be used with the GROUP BY clause of SQL. For more information, see Aggregate functions.
UDTF	User-defined table-valued function. When a UDTF is called, it generates multiple columns or rows of data.

Example

Realtime Compute for Apache Flink provides an example of a UDX to facilitate your business development. This example shows how to develop a UDF, UDAF, and UDTF.

> **Note**
>
> - In this example, a development environment of the required version is configured. You do not need to build another development environment.
> - The example provides Maven projects. You can use IntelliJ IDEA for development. For more information, see Develop a job.

- Realtime Compute for Apache Flink V3.0

  Blink_UDX_3x
- Realtime Compute for Apache Flink V2.0

  Blink_UDX_2x
- Realtime Compute for Apache Flink V1.0

  Blink_UDX_1x

Build a development environment

The development of UDXs depends on the following JAR packages. You can download the packages as required.

- Realtime Compute for Apache Flink versions earlier than V3.2.1
  - flink-streaming-java_2.11
  - flink-table_2.11
  - flink-core-blink-2.2.4
- Realtime Compute for Apache Flink V3.2.1 and later

  Add a POM dependency based on your open source software version. Download and view the example of a complete dependency.

> **Note**
>
> If you want to use Snapshot, you can add a POM dependency based on your Snapshot version.

Register and use resources

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure the resource parameters.

Parameter	Description
Location	You can upload JAR packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Select Resource Name	Enter a name for the resource.
Resource Description	Enter a description for the resource.
Authorization type	Select the type of the resource: JAR, DICTIONARY, or PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.

7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX at the beginning. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Types of parameters and return values

When you define Java UDXs in Realtime Compute for Apache Flink, you can use Java data types in parameters and return values. The following table lists the mappings between Realtime Compute for Apache Flink and Java data types.

Data type of Realtime Compute for Apache Flink	Java data type
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL	java.math.BigDecimal
BOOLEAN	java.lang.Boolean
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CHAR	java.lang.Character
STRING	java.lang.String
VARBINARY	java.lang.byte[]

ARRAY	Not supported
MAP	Not supported

Obtain parameters of UDXs

UDXs support an optional `open(FunctionContext context)` method. You can use `FunctionContext` to pass custom configuration items.

For example, you must add the following two parameters to your job:

```
testKey1=lincoln
test.key2=todd
```

The following example shows how to use `context.getJobParameter` in the open method to obtain parameters of a UDTF.

```
public void open(FunctionContext context) throws Exception {
      String key1 = context.getJobParameter("testKey1", "empty");
      String key2 = context.getJobParameter("test.key2", "empty");
      System.err.println(String.format("end open: key1:%s, key2:%s", key1, key2));
}
```

> **Note**
> For more information, see Job parameters.

5.11.2. UDF

This topic describes how to build a development environment, write business logic code, and publish a user-defined scalar function (UDF) in Realtime Compute for Apache Flink.

Definition

A UDF maps zero, one, or more scalar values to a new scalar value.

Build a development environment

For more information about how to build a development environment, see Build a development environment.

Write business logic code

To define a UDF, you must extend the ScalarFunction class by implementing the `eval` method. The `open` and `close` methods are optional.

> **Important**
>
> UDFs return the same output for the same input by default. However, a UDF where an external service is called may return different output results even if the input values are the same. If a UDF cannot generate the same output for the same input, we recommend that you use the `override isDeterministic()` method to make it return `false`. Otherwise, the output may not meet your expectations in some cases. For example, a UDF operator moves forward.

The following sample code is written in Java:

```
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 use the open method, you must add 'import
org.apache.flink.table.functions.FunctionContext;' to the code.
    @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() {
        }
}
```

Write SQL statements

You can write SQL statements in a specified class. The following example shows the SQL statements in a UDX:

```
-- 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;
```

Register and use resources

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure the resource parameters.

Parameter	Description

Location	You can upload only JAR packages from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the size of the JAR package exceeds 300 MB, you must upload the package to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload the package.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a description for the resource.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.
7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX at the beginning. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Publish and use a UDF

Click **Publish** on the **Development** page for the job where you want to publish a UDF. Then, find the job on the **Administration** page and click **Start** in the Actions column to publish the UDF.

FAQ

Q: Why does the random number generator always generate the same value at runtime?

A: If no input parameters are passed to a UDX and you do not declare it as nondeterministic, the UDX may be optimized during compilation to return a constant value. To avoid this issue, you can use the `override isDeterministic()` method to make it return `false`.

5.11.3. UDAF

This topic describes how to build a development environment, write business logic code, and publish a user-defined aggregation function (UDAF) in Realtime Compute for Apache Flink.

Definition

A UDAF aggregates multiple values into a single value.

Methods of the UDAF abstract class

> **Note**
>
> A UDAF can be implemented in Java or Scala. However, we recommend that you use Java because Scala data types may cause unnecessary performance overhead.

The following code shows the core methods of the AggregateFunction class.

- createAccumulator and getValue methods

```
/*
* @param <T> The type of the output returned by a UDAF.
* @param <ACC> The accumulator type of a UDAF. An accumulator stores the intermediate
aggregation results of a UDAF. You can design an accumulator for each UDAF as require
d.
*/
public abstract class AggregateFunction<T, ACC> extends UserDefinedFunction {
/*
* Initialize the accumulator in AggregateFunction.
* The system calls the following method before it aggregates data for the first time.

*/
public ACC createAccumulator();
/*
* The system 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 input]...);
```

> **Note**
>
> - You must implement an accumulate method to describe how to compute input data and update an accumulator to the aggregation result.
> - The first parameter of the accumulate method must be an accumulator of the ACC type defined in AggregateFunction. When the system is running, the runtime code sends the previous value in the accumulator and the specified upstream data to the accumulate method for aggregation. The upstream data can be of any type and can contain any number of data records.

- retract and merge methods

The createAccumulator, getValue, and accumulate methods can be used together to design a basic UDAF. However, Realtime Compute for Apache Flink also requires the retract and merge methods in some special scenarios.

In most scenarios, computing is early firing for an infinite stream. To refine early fired results, you can implement a retract method. The SQL optimizer automatically determines the conditions in which data needs to be retracted and the operations that are needed to process data marked with retract tags. You must implement a retract method to retract input data.

```
public void retract(ACC accumulator, ...[user input]...);
```

> **Note**
>
> - The retract method is the reverse operation of the accumulate method. For example, in a count UDAF, the number of data records in the computing result increases by one each time the accumulate method is called to process a data record, whereas the number of data records in the result decreases by one each time 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 defined in AggregateFunction. When the system is running, the runtime code sends the previous value in the accumulator and the specified upstream data to the retract method for aggregation. The upstream data can be of any type and contain any number of data records.

Realtime Compute for Apache Flink requires the merge method in some scenarios. For example, if you use a session window to aggregate data, you must use the merge method. Realtime Compute for Apache Flink can process 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 must use 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 defined in AggregateFunction. After the merge method is executed, the state data of AggregateFunction is stored in the first accumulator.
> - The second parameter of the merge method is an iterator of one or more accumulators of the ACC type.

Build a development environment

For more information about how to build a development environment, see Build a development environment.

Write business logic code

The following Java code is an example:

```
import org.apache.flink.table.functions.AggregateFunction;

public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
    // Define the data schema of the accumulator that stores the state data of a count
UDAF.
    public static class CountAccum {
        public long total;
    }

    // Initialize the accumulator of the count UDAF.
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }

    // Call the getValue method to obtain the result of the count UDAF from the accumul
ator that stores the state data of the count UDAF.
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }

    // Call the accumulate method to update the accumulator that stores the state data
of the count UDAF based on the 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**
>
> The open and close methods are optional for a subclass of AggregateFunction. For more information, see the examples of UDF or UDTF.

Register and use resources

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure resource parameters.

Parameter	Description

Location	You can upload Java Archive (JAR) packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a resource description.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.
7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Publish and use a UDAF

For more information about how to publish and use a UDAF, see Publish a job and Start a job.

Example

```
-- Use 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='<yourPorjectName>',
   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;
```

5.11.4. UDTF

This topic describes how to build a development environment, write code, and publish a user-defined table-valued function (UDTF) in Realtime Compute for Apache Flink.

Definition

Similar to a user-defined scalar function (UDF), a UDTF uses zero, one, or multiple scalar values as input parameters (including variable-length parameters). Different from a UDF, a UDTF returns any number of rows, rather than a single value. The returned rows can consist of one or more columns.

Build a development environment

For more information, see Build a development environment.

Write code that implements business logic

A UDTF needs to implement the eval method in the TableFunction class. The open and close methods are optional. The following Java code is an example:

```
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 use the open method, you must add 'import org.ap
ache.flink.table.functions.FunctionContext;' to the code.
    @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 convert the output result from a single row to multiple rows by calling the `collect` method multiple times.

Return multiple columns

A UDTF can also convert the output result from a single column to multiple columns. If you want a UDTF to return multiple columns, declare the return value as a tuple or row. The following examples show how to declare a return value as a tuple and how to declare a return value as a row.

- Declare the return value as a tuple.

  Realtime Compute for Apache Flink supports Tuple1 to Tuple25, which define 1 to 25 fields. The following example is a UDTF that uses Tuple3 to return three fields:

```
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.table.functions.TableFunction;

// If the return value is declared as a tuple, you must explicitly declare the generic types of the tuple, such 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 reference only. In actual scenarios, you must add code that implements 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, column values cannot be null and a maximum of 25 columns are allowed.

- Declare the return value as a row.

  For example, you can enable this feature to return three columns. The sample code is an example.

```
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, you must overload the getResultType meth
od to explicitly declare the data type of the return value.
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, the field value can be null. However, you must overload the `getResultType` method.

SQL syntax

A UDTF supports CROSS JOIN and LEFT JOIN. When you use a UDTF, you must add the keywords `LATERAL` and `TABLE`. You must also specify an alias for the UDTF, such as `ParseUdtf` in the preceding code.

```
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**
>
> A LEFT JOIN statement that uses a UDTF must end with `on true` .

```
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;
```

Register and use resources

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure resource parameters.

Parameter	Description
Location	You can upload JAR packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a resource description.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.
7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX at the beginning. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Publish and use a UDTF

For more information about how to publish and use a UDTF, see Publish a job and Start a job.

Example

```
-- 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. Table T(s) with multipl
e rows and one column is returned. s is the name of the column.
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;
```

5.11.5. Develop a UDX by using IntelliJ IDEA

This topic describes how to develop a user-defined extension (UDX) in Realtime Compute for Apache Flink by using IntelliJ IDEA. The development process includes building a development environment and referencing a UDX in a Realtime Compute for Apache Flink job.

Background information

> **Important**
> - Only Realtime Compute for Apache Flink in exclusive mode supports UDXs.
> - We recommend that you use IntelliJ IDEA to develop a UDX.

Configure Maven

1. Download a Maven installation package.
   i. Go to the download page at the Apache Maven official website.
   ii. Download **apache-maven-3.5.3-bin.tar.gz**.
   iii. Decompress the downloaded package to a specified directory, such as /Users/<userName>/Documents/maven.
2. Configure environment variables.
   i. In the command terminal, run the `vim ~/.bash_profile` command.
   ii. Add the following commands to the **.bash_profile** file:

   ```
   export M2_HOME=/Users/<userName>/Documents/maven/apache-maven-3.5.3
   export PATH=$PATH:$M2_HOME/bin
   ```

   iii. Save and close the **.bash_profile** file.
   iv. Run the `source ~/.bash_profile` command for the configuration to take effect.
3. Run the `mvn -v` command to check whether the configuration takes effect.
   If information similar to the following information is displayed, the configuration takes effect:

   ```
   Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-04T03:39:06+08:00)
   Maven home: /Users/<userName>/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"
   ```

Build a development environment

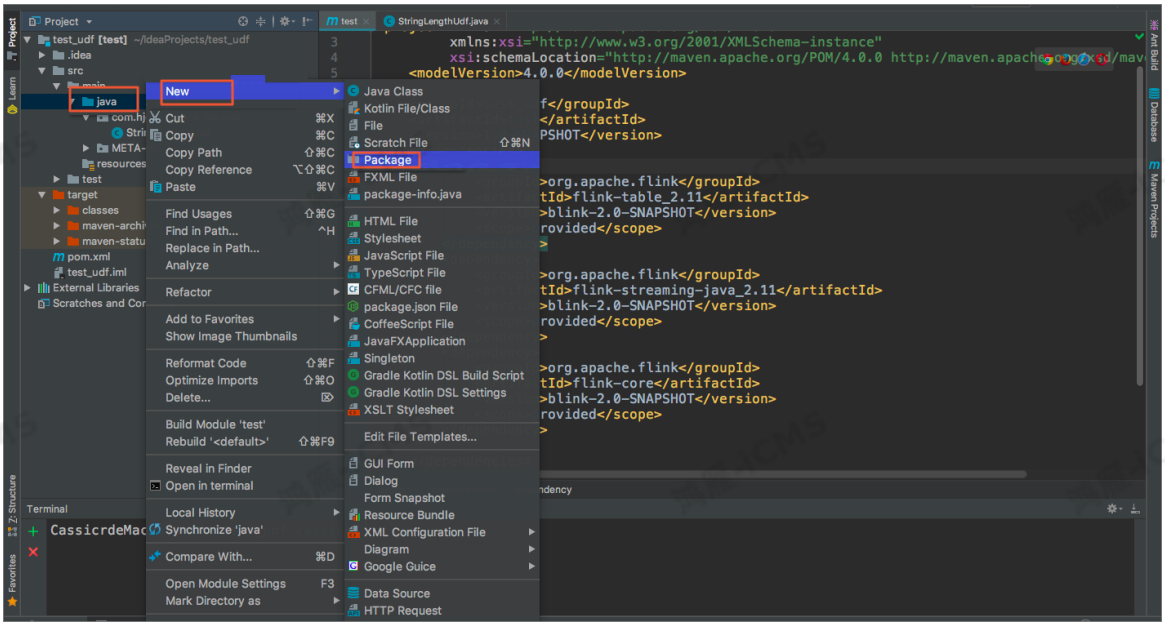
1. Download the UDX demo.
2. Decompress the downloaded package in a Linux operating system.

   ```
   tar xzvf RealtimeCompute-udxDemo.gz
   ```

3. Open **IntelliJ IDEA** and click **Open** to open the demo.

Reference a JAR package in a job

1. Create a package.
   i. Right-click **java** and choose **New > Package**.



   ii. In the **New Package** dialog box, enter a package name. In this example, a package named `com.hjc.test.blink.sql.udx` is created.
   iii. Click **OK**.
2. Create a class.
   i. Right-click **com.hjc.test.blink.sql.udx** and choose **New > Java Class**.

ii. In the **Create New Class** dialog box, enter a class name.

> **Note** Retain the default value **Class** of **Kind**.

iii. Click **OK**.

3. Enter the following code in the class:

```
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 add 'import
org.apache.flink.table.functions.FunctionContext;' to the code.
    @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. In the command terminal, run the `mvn package` or `mvn assembly:assembly` command to add the project to the JAR package.

> **Note**
> - If you need to add a third-party dependency to the JAR package, run the `mvn assembly:assembly` command.
> - The compiled JAR package is **RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT.jar** or **RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT-jar-with-dependencies.jar**. It contains a third-party dependency.

5. Reference the JAR package in a Realtime Compute for Apache Flnik job.

i. Log on to the Realtime Compute development platform.

ii. In the top navigation bar, click **Development**.

iii. In the left-side navigation pane, click the **Resources** tab.

iv. In the upper-right corner of the **Resources** pane, click **Create Resource**.

Parameter	Description
Location	You can upload JAR packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a resource description.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

v. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.

vi. In the drop-down list, select **Reference**.

vii. In the code editor, declare the UDX at the beginning. The following code is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

6.Blink SQL Development Guide

6.1. Overview

The Realtime Compute for Apache Flink development platform provides multiple features for Realtime Compute Flink SQL jobs, including data storage management, job development, job debugging, job administration, monitoring and alerting, and job optimization.

Flink SQL Developer Guide consists of the following topics:

- Data storage

  You can manage upstream and downstream data storage systems, such as ApsaraDB RDS, DataHub, and Tablestore, for jobs on the Realtime Compute for Apache Flink development platform. After you register resources from these systems with Realtime Compute for Apache Flink, you can preview or sample their related data, or obtain the data definition language (DDL) statements that are automatically generated to reference these resources. For more information about data storage, see Overview.

  > **Note** For more information about how to add the IP addresses of Realtime Compute for Apache Flink to a whitelist of an upstream or downstream storage system, see Configure a whitelist for accessing storage resources.

- Job development

  This topic describes how to develop, publish, and start a Flink SQL job. For more information, see Develop a job, Publish a job, and Start a job.

- Job debugging

  This topic describes how to debug Flink SQL jobs. Online debugging and local debugging are supported. For more information, see Online debugging.

- Job administration

  This topic describes how to view the administration information of a Realtime Compute for Apache Flink job, such as the running information, curve charts, and failover. For more information, see Overview, Metrics, and Failover.

- Monitoring and alerting

  This topic describes how to create and activate alert rules. For more information, see Monitoring and alerting.

- Job optimization

  This topic describes how to optimize Flink SQL jobs, such as skills for optimizing Flink SQL code, automatic configuration optimization, performance optimization by auto scaling, and performance optimization by manual configuration. For more information, see Recommended Flink SQL practices,Performance optimization by using auto scaling, and Optimize performance by manual configuration.

- Flink SQL

  This topic describes the syntax of Flink SQL. For more information, see Overview.

6.2. Data storage

6.2.1. Overview

Alibaba Cloud Realtime Compute for Apache Flink provides a page to manage various storage systems, such as ApsaraDB RDS and Tablestore. Realtime Compute for Apache Flink provides you an end-to-end cloud-based management solution.

Limits

A Realtime Compute for Apache Flink cluster in exclusive mode can access only storage resources in the same virtual private cloud (VPC), region, and security group as the cluster.

Data storage in Realtime Compute for Apache Flink

In Realtime Compute for Apache Flink, data storage has the following meanings:

- It refers to the storage systems or database tables (hereinafter referred to as storage resources) at the upstream and downstream nodes of Realtime Compute for Apache Flink.
- It indicates how to use the data storage feature of Realtime Compute for Apache Flink. This feature is used to manage the upstream and downstream storage resources.

> **Note** Before you register storage resources with Realtime Compute for Apache Flink, you must authorize Realtime Compute for Apache Flink to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

Realtime Compute for Apache Flink allows you to reference both upstream and downstream storage resources by using plaintext AccessKey pairs or registering storage resources.

Use a plaintext AccessKey pair

To reference upstream and downstream storage resources by using a plaintext AccessKey pair, you must configure the `accessId` and `accessKey` parameters in the WITH clause of the related DDL statement. For more information, see Overview. This way, you can authorize an Alibaba Cloud account and its RAM users to access the resources of the current or another Alibaba Cloud account. If User A or a RAM user created within the Alibaba Cloud account of User A wants to use the storage resources of User B, User A can set the AccessKey pair of User B in the following DDL statement in plaintext mode:

```
CREATE TABLE in_stream(
  a varchar,
  b varchar,
  c timestamp
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyuncs.com',
  project='<dataHubProjectName>',
  topic='<dataHubTopicName>',
  accessId='<accessIdOfUserB>',
  accessKey='<accessKeyOfUserB>'
);
```

Register a storage resource

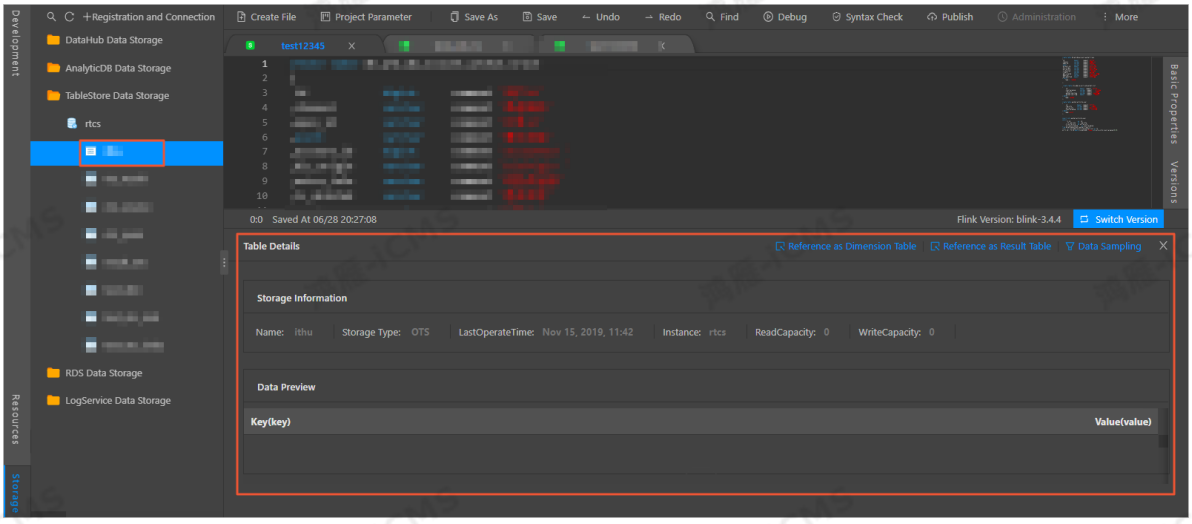
Realtime Compute for Apache Flink allows you to manage and reference both upstream and downstream storage resources that have been registered with Realtime Compute for Apache Flink. After storage resources are registered, you can preview or sample the relevant data, or obtain the DDL statements that are automatically generated to reference the resources. This helps you manage your cloud storage resources in end-to-end mode.

> **Note** You can register only storage resources of the current Alibaba Cloud account. Therefore, User A or a RAM user created within the Alibaba Cloud account of User A can register only storage resources purchased by User A. If you want to use storage resources of another Alibaba Cloud account, you must use the plaintext AccessKey pair of the specified Alibaba Cloud account in the relevant DDL statement.

- Register storage resources
  To register upstream and downstream storage resources with Realtime Compute for Apache Flink before you reference them, perform the following steps:
  i. Log on to the Realtime Compute development platform.
  ii. In the top navigation bar, click **Development**.
  iii. In the left-side navigation pane of the **Development** page, click **Storage**.
  iv. In the upper-right corner of the **Storage** tab, click **+Registration and Connection**.
  v. In the **Register Data Store and Test Connection** dialog box, configure the parameters for storage resources.

  Realtime Compute for Apache Flink allows you to register the following types of storage resources. For more information about how to register storage resources of a specific type, click the following links:

  - Register a Tablestore instance
  - Register an ApsaraDB for RDS instance
  - Register a Log Service project
- Preview data from a registered storage resource
  To preview data from a registered storage resource, perform the following steps:
  i. In the left-side navigation pane of the **Development** page, click **Storage**.
  ii. On the **Storage** tab, double-click the folder of a registered storage resource and its subfolder to find the table that you want to view, and double-click the name of the table.
  iii. In the **Table Details** pane, view data of the storage resource in the **Data Preview** section.

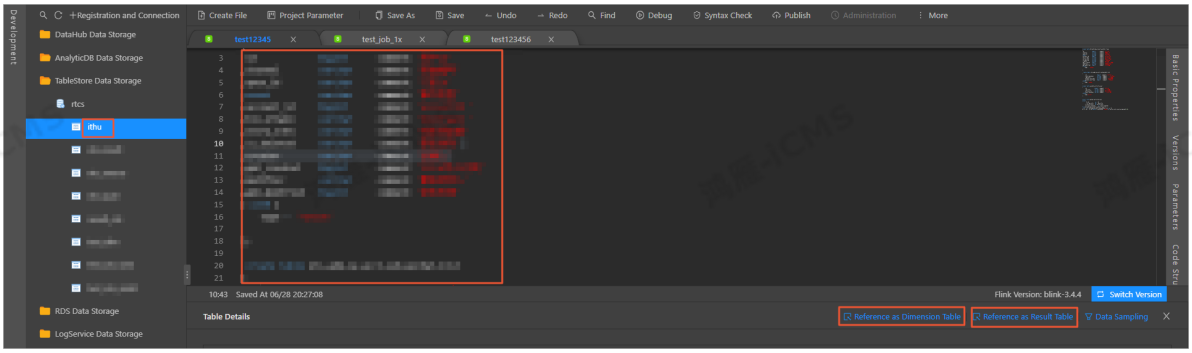


- Obtain the DDL statements that are automatically generated to reference a storage resource
  To obtain the DDL statements that are automatically generated to reference a storage resource, perform the following steps:
  i. In the left-side navigation pane of the **Development** page, click **Storage**.
  ii. On the **Storage** tab, double-click the folder of a registered storage resource and its

subfolder to find the table that you want to view, and double-click the name of the table.

iii. In the **Table Details** pane, click **Reference as Source Table**, **Reference as Result Table**, or **Reference as Dimension Table**. Then, you can obtain the DDL statements that are automatically generated to reference the table.

> **Note** The automatically generated DDL statements contain only the basic parameters in the WITH clause to ensure connectivity between Realtime Compute for Apache Flink and storage resources. You can add other parameters to the WITH clause in addition to the basic parameters.



- Test network connectivity by using the network detection feature

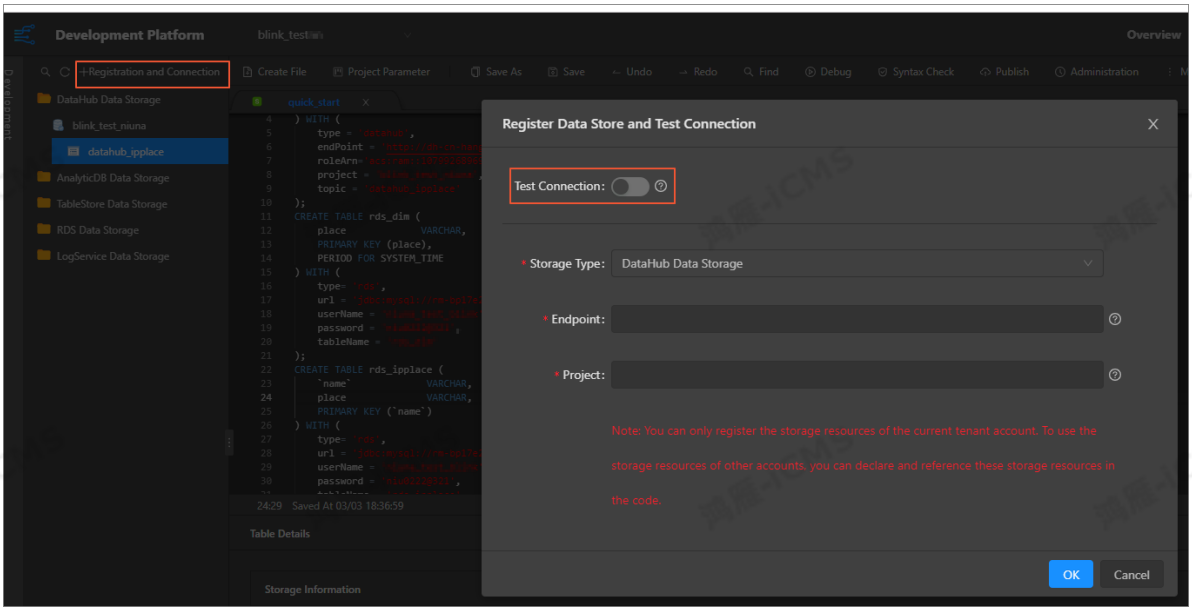
> **Note** The network detection feature is not supported in the China (Hangzhou) region of Finance Cloud because Cloud Assistant is not installed in the region.

Realtime Compute for Apache Flink provides the network detection feature for data storage. This feature allows you to test network connectivity between Realtime Compute for Apache Flink and storage resources. To enable the network detection feature, perform the following steps:

i. In the left-side navigation pane of the **Development** page, click **Storage**.

ii. In the upper-right corner of the **Storage** tab, click **+Registration and Connection**.

iii. In the **Register Data Store and Test Connection** dialog box, turn on **Test Connection**.



6.2.2. Data storage resource registration

6.2.2.1. Register an AnalyticDB for MySQL instance

This topic describes the parameters that are required to register an AnalyticDB for MySQL instance.

> **Important**
>
> This topic applies only to AnalyticDB for MySQL V2.0. Realtime Compute for Apache Flink does not allow you to register AnalyticDB for MySQL V3.0 to store result tables. To use the result tables of AnalyticDB for MySQL V3.0, you must create and reference the result tables in plaintext mode. For more information, see Create an AnalyticDB for MySQL V3.0 result table.

Register storage resources

> **Note**
>
> Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.
5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.
6. Click **OK**.

Storage parameters

Parameter	Description
URL	Enter the VPC URL of the AnalyticDB for MySQL database To view the URL, perform the following steps: 1. Log on to the AnalyticDB for MySQL console. 2. In the left-side navigation pane, click Clusters. On the Clusters page, click the ID of the instance to go to the **Cluster Information** page. 3. In the **Network Information** section, view the URL.

Database	The name of the AnalyticDB for MySQL database or the name of the AnalyticDB for MySQL instance.
AccessKey ID	The AccessKey ID of your Alibaba Cloud account.
AccessKey Secret	The AccessKey secret of your Alibaba Cloud account.

6.2.2.2. Register a Tablestore instance

This topic describes how to use Realtime Compute for Apache Flink to register a Tablestore instance.

Introduction to Tablestore

Tablestore is a NoSQL database service that is built based on Alibaba Cloud Apsara system. Tablestore allows you to store and access large amounts of structured data in real time. Tablestore provides the following benefits: low latency and simple computing. Therefore, Tablestore is suitable for storing dimension tables and result tables of Realtime Compute for Apache Flink.

Register storage resources

> **Note**
>
> Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.
5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.
6. Click **OK**.

Parameters

- Endpoint
  - The virtual private cloud (VPC) endpoint of the Tablestore instance that you want to register. You can view this endpoint in the Tablestore console. For more information about the VPC endpoint, see Endpoints.

- To set Accessed By to **Any Network**, perform the following steps:
  a. Log on to the Tablestore console.
  b. In the left-side navigation pane, click All Instances. In the **Instance Name** column of the All Instances page, click the name of the Tablestore instance that you want to register.
  c. On the Instant Management page, click the **Network Management** tab. On this tab, click **Change** next to Accessed By.
  d. In the dialog box that appears, select **Any Network** from the Accessed By drop-down list.
  e. Click **OK**.

- Instance Name

  Enter the name of the Tablestore instance.

6.2.2.3. Register an ApsaraDB for RDS instance

This topic describes how to use Realtime Compute for Apache Flink to register an ApsaraDB for RDS instance.

Introduction to ApsaraDB for RDS

ApsaraDB for RDS is a stable, reliable, and scalable online database service. Based on Apsara Distributed File System and high-performance storage services, ApsaraDB for RDS supports a wide range of database engines, such as MySQL, SQL Server, PostgreSQL, and Postgres Plus Advanced Server (PPAS). ApsaraDB for RDS provides comprehensive solutions for database operations and maintenance (O&M), such as disaster recovery, data backup, data recovery and restoration, monitoring, and data migration.

> **Note**
> - If Realtime Compute for Apache Flink frequently writes data to a table or a resource file, a deadlock may occur. In scenarios that require highly frequent or highly concurrent writes, we recommend that you use Tablestore to store result tables. For more information, see Create a Tablestore result table.
> - Realtime Compute for Apache Flink does not allow you to register the ApsaraDB for RDS V8.0 data store in the console to store result tables. To use result tables of ApsaraDB for RDS V8.0, you must use the plaintext mode to create and reference the result tables. For more information, see Use a plaintext AccessKey pair.

Register storage resources

> **Note** Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.
5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.

6. Click **OK**.

Parameters

> **Note** When you register storage resources in an ApsaraDB for RDS instance, the IP addresses of Realtime Compute for Apache Flink are automatically added to the whitelist for accessing the ApsaraDB for RDS instance.

Parameter	Description
Storage Type	The type of storage resources that you want to register. From the Storage Type drop-down list, select **RDS Data Storage**.
Region	The region where the ApsaraDB for RDS instance resides.
Instance	The ID of the ApsaraDB for RDS instance that you want to register. **Note** Enter the instance ID instead of the instance name.
DBName	The name of the ApsaraDB for RDS database. **Note** Enter the database name instead of the instance name.
User Name	The username that is used to access the ApsaraDB for RDS database.
Password	The password that is used to access the ApsaraDB for RDS database.

6.2.2.4. Register a Log Service project

This topic describes how to use Realtime Compute for Apache Flink to register a Log Service project. This topic also provides answers to commonly asked questions about the registration process.

Introduction to Log Service

Log Service is an end-to-end logging service. Log Service allows you to collect, consume, ship, query, and analyze log data in a quick way. It improves the operations and maintenance (O&M) efficiency, and provides the capability to process large amounts of data. Log Service is used to store streaming data. Therefore, Realtime Compute for Apache Flink can use the streaming data that is stored in Log Service as input data.

Register storage resources

> **Note**
>
> Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.
5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.
6. Click **OK**.

Parameters

- Endpoint

  The endpoint of the Log Service project that you want to register. The endpoint of a Log Service project varies based on the region of this project. For more information, see Endpoints.

  > **Note**
  > - The endpoint of a Log Service project must start with `http://` and cannot end with a forward slash (`/`). For example, the endpoint can be `http://cn-hangzhou-intranet.log.aliyuncs.com` .
  > - Realtime Compute for Apache Flink and Log Service are deployed in the internal network of Alibaba Cloud. We recommend that you enter the endpoint of the classic network or the virtual private cloud (VPC) for the project. We recommend that you do not enter the public endpoint. If Realtime Compute for Apache Flink accesses a Log Service project over the Internet by using the public endpoint, the system may consume the resources of Internet bandwidth. In this case, the system performance may be compromised.

- Project

  The name of the Log Service project that you want to register.

  > **Note**
  > In Realtime Compute for Apache Flink, you can register only the Log Service projects that are owned by the current Alibaba Cloud account. Assume that User A owns Project A of Log Service. If User B needs to use the storage resources of Project A in Realtime Compute for Apache Flink, this system does not allow User B to register Project A. If you need to use the Log Service project that is owned by another Alibaba Cloud account, you can use the plaintext mode to use the project. For more information, see Use a plaintext AccessKey pair.

FAQ

What do I do if I fail to register a storage resource in Realtime Compute for Apache Flink?

Realtime Compute for Apache Flink uses a storage software development kit (SDK) to access different data storage systems. The Storage tab on the Realtime Compute for Apache Flink development platform helps you manage data from different data storage systems. If you fail to register a storage resource in Realtime Compute for Apache Flink, troubleshoot the issue by performing the following steps:

- Check whether the Log Service project is created in your Alibaba Cloud account. Log on to the Log Service console and check whether you can access the project.

- Check whether the Log Service project is owned by your Alibaba Cloud account. You cannot register a project that is owned by another Alibaba Cloud account.
- Check whether the endpoint and the name of the Log Service project are valid. The endpoint of the Log Service project must start with `http://` and cannot end with a forward slash (`/`).
- Check whether the endpoint of the Log Service project is the classic network endpoint. Realtime Compute for Apache Flink does not support VPC endpoints.
- Check whether you have already registered the Log Service project. Realtime Compute for Apache Flink provides a registration check mechanism to prevent duplicate registrations.

Why does Realtime Compute for Apache Flink support only time-based data sampling?

Log Service stores streaming data and supports only time-based data sampling. You can specify only time parameters in the Log Service API. Therefore, Realtime Compute for Apache Flink supports only time-based data sampling.

6.2.3. Configure a whitelist for accessing storage resources

By default, a newly created database instance does not allow access from IP addresses that are not included in its whitelists. To allow Realtime Compute for Apache Flink to access the database instance, you must add the IP addresses of Realtime Compute for Apache Flink to a whitelist of the database instance. This topic describes how to add the IP addresses of Realtime Compute for Apache Flink to a whitelist of an ApsaraDB for RDS instance.

IP addresses to be added to the whitelist

To access storage resources from a Realtime Compute for Apache Flink cluster in exclusive mode, you only need to add the IP addresses of the ENI to the whitelist. To view the IP addresses of the ENI, perform the following steps:

1. Log on to the .
2. Move the pointer over the username in the upper-right corner.
3. In the drop-down list, click **Project Management**.
4. In the left-side navigation pane, click **Clusters**.
5. On the **Clusters** page, click the name of the target cluster.
6. In the cluster information dialog box, view the **ENI** of the cluster.

Configure a whitelist for an ApsaraDB for RDS instance

When you reference an ApsaraDB for RDS database in Realtime Compute for Apache Flink, Realtime Compute for Apache Flink needs to frequently read and write data in the ApsaraDB for RDS database. In this case, you must add the IP addresses of Realtime Compute for Apache Flink to a whitelist of the ApsaraDB for RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for MySQL instance.

6.3. Job development

6.3.1. Develop a job

This topic describes how to create a Realtime Compute for Apache Flink job. This topic also describes the features provided on the Development page, such as syntax check, Flink SQL code assistance, and management of Flink SQL code versions.

Background information

> **Note**

Write the Flink SQL code of a job

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. At the top of the **Development** page, click **Create File**
4. In the **Create File** dialog box, specify the parameters. The following table describes these parameters.

Parameter	Description
File Name	The name of the file that contains the Fink SQL code. **Note** The file name must be unique in the existing project.
File Type	Valid values are **FLINK_STREAM/DATASTREAM** and **FLINK_STREAM/SQL**.
Storage Path	The folder of the file that contains the Fink SQL code. On the right side of the existing folder, you can also click the  icon to create a subfolder.

5. Click **OK**.
6. On the page that appears, write the Flink SQL code for the job.

> **Note**
>
> - On the right side of the **Development** page, you can click **Code Structure** to check the Flink SQL code structure.
> - On the left side of the **Development** page, we recommend you click **Storage** to manage upstream and downstream storage resources. For more information, see Overview.

Specify job parameters

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the **Development** page, click the job name.
4. On the right side of the **Development** page for the job, click **Parameters**.
5. Specify the parameters required for the job.

   For more information about job parameters, see Job parameters.

Specify project parameters

The job parameters are valid for a single job. The project parameters are valid for all the jobs in the project. After the project parameters are configured, the following two operations are performed:

- Replacing variables: After you click Start, Debug, or Syntax Check, the system replaces the variables in SQL jobs or in the code of the jobs that are created by using the Flink DataStream API.
- Distributing parameters: The project-level system parameters are merged with job parameters and startup parameters. The startup parameters can be configured for only batch jobs. The following parameters are sorted in descending order based on their priorities: startup parameters > job parameters > project-level system parameters. The merged parameters are used as final parameters and are distributed to the Blink job. For example, if job parameters and project parameters conflict, the job parameters prevail.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, move the pointer over your profile picture. In the list that appears, click **Project Management**.
3. In the **Project Name** column of the **Projects** page, click the name of the project for which you want to configure parameters.
4. In the top navigation bar, click **Development**.
5. On the left side of the **Development** page, click the job for which you want to configure the parameters.
6. Enable project parameter configuration.

   By default, `disable.project.config=false` is specified. This indicates that you cannot configure project parameters. You can enable project parameter configuration by using the following methods:

   - SQL jobs: On the right side of the Development page, click **Parameters** and specify the following setting: `enable.project.config=true` .
   - Flink DataStream job: In the job code, specify `enable.project.config=true parameter` .
7. At the top of the Development page, click **Project Parameter**.
8. Configure the project parameters

   Project parameters support only two job types: SQL jobs and Flink DataStream jobs. When you configure project system parameters, you must add the job type to the beginning of the project parameters. For example, you can use `sql.name=LiLei` or `datastream.name=HanMeimei` .

Enable syntax check

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. On the left side of the **Development** page, click the job for which you want to check syntax.
4. At the top of the **Development** page, click **Syntax Check**.

> **Note**
>
> - When you save a Flink SQL job, the system automatically checks the syntax of this job.
> - **Syntax Check** takes effect for only Flink SQL statements that have complete logic. Otherwise, **Syntax Check** does not take effect.

Flink SQL code assistance

- Syntax check

  After you modify the Flink SQL code, the system automatically saves the code and checks the syntax. If a syntax error is detected, the system displays the cause of the error, the row and column where the error occurred on the **Development** page.

- Intelligent code completion

  When you enter Flink SQL statements on the **Development** page, the system automatically performs intelligent code completion, including keywords, built-in functions, tables, and fields.

- Syntax highlighting

  The system highlights keywords in Flink SQL statements and displays different structures in different colors.

Management of Flink SQL code versions

Realtime Compute for Apache Flink allows you to manage Flink SQL code versions. A new code version is generated each time you publish a job. You can use code versions to track versions, modify the code, and roll the code back to an earlier version.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. On the left side of the **Development** page, click the job for which you want to manage the code version.
4. On the right side of the **Development** page, click **Versions**.
5. In the Versions pane, find the code version of the job and choose **Actions > More**.
6. Select one of the following options from the drop-down list:
   - **Compare**: checks the difference between the current version and an earlier version.
   - **Rollback**: rolls the code back to an earlier version.
   - **Delete**: deletes a code version. By default, you can reserve a maximum of 20 Flink SQL code versions in Realtime Compute for Apache Flink. If the number of code versions is less than 20, you can publish a job. If the number of code versions is 20, the system does not allow you to publish a job and prompts you to delete earlier versions.

     > **Note**
     >
     > You can publish a job only if the number of versions is less than 20.

   - **Locked**: locks the current version.

     > **Note**
     >
     > You can submit a new version only after you unlock the current version.

6.3.2. Publish a job

After you develop and debug a job, and pass the syntax check, you can publish the job to the production environment.

Procedure

1. Configure resources
   Specify the resource configuration mode based on your requirements. We recommend that you use the default configuration if you publish the job for the first time.

   > **Note** Realtime Compute for Apache Flink supports manual resource configuration. For more information, see Optimize performance by manual configuration.

2. Check data

   Check parameter settings and click **Next**.

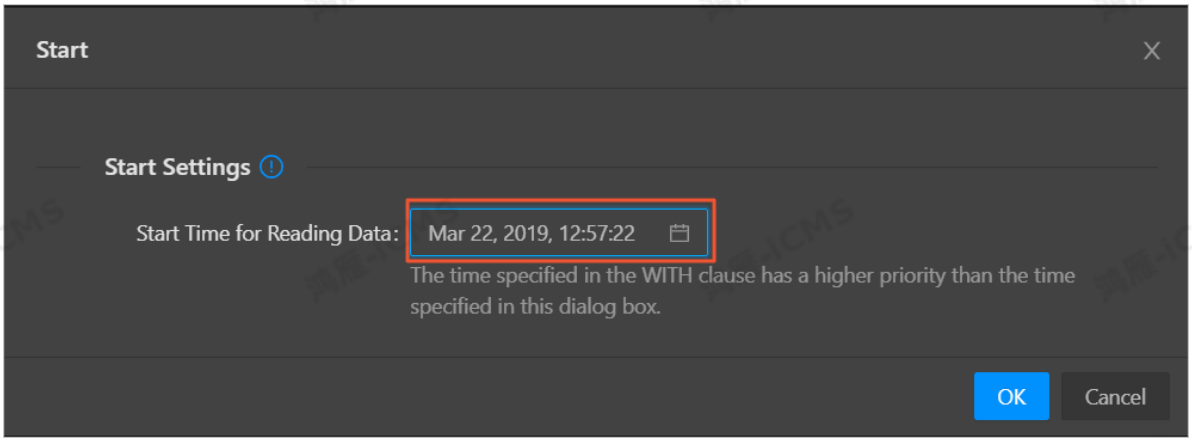
3. Publish the job
   Click **Publish**.

6.3.3. Start a job

After you develop and publish a job, you can start the job on the Administration page.

Procedure

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration**.
3. In the **Actions** column of the **Administration** page, find the job that you want to start and click **Start**.
4. In the **Start** dialog box, configure the **Start Time for Reading Data** parameter.



5. Click **OK**. The job is started.
   Start Time for Reading Data specifies the time when Realtime for Apache Flink starts to read data from the source table. The time indicates the time when data is generated.
   - If you specify the current time, Realtime Compute for Apache Flink reads data that is generated from the current time.
   - If you select a previous time point, Realtime Compute for Apache Flink reads data that is generated from this time point. This is used to trace historical data.

> **Note** After the job is started, you can check the running information on the Overview tab. For more information, see Overview.

6.3.4. Suspend a job

After you modify the resource configuration of a job, you must suspend and resume the job to make the changes take effect. This topic describes how to suspend a job.

Background information

> **Important**
> - You can only **suspend** a job that is in the **Running** state.
> - **Suspending** a job does not clear its task status. For example, if the job you **suspend** is running a COUNT operation, the COUNT operation continues from the last successful checkpoint after you **resume** the job.
> - The Suspend (checkpoint) operation is supported in Realtime Compute V3.5.0 and later. If your Realtime Compute is earlier than V3.5.0, the following error message is displayed when you try to perform this operation: **An error occurred. System error: The BLINK version is abnormal. Error reason: blink version >= blink-3.5 is required, instance blink-3.4.4**.

Procedure

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration**.
3. On the **Administration** page, find the target job, and click **Suspend** in the **Actions** column.

> **Note** The **Suspend (checkpoint)** operation in **More** suspends the job and triggers a checkpoint event. Therefore, the **Suspend (checkpoint)** operation takes longer time to suspend a job than the **Suspend** operation.

6.3.5. Terminate a job

After you modify the SQL logic, change the job version, add parameters to the WITH clause, or add job parameters for a job, you must terminate and then start the job to make the changes take effect. This topic describes how to terminate a job.

> **Important**
> - You can only **terminate** a job that is in the **Running** or **Starting** state.
> - If you **terminate** a job, its task status is cleared. For example, if the job you **terminate** is running a COUNT operation, the COUNT operation starts from 0 after you **start** the job.
> - The **Terminate (checkpoint)** operation is supported in Realtime Compute for Apache Flink V3.5.0 and later. If your Realtime Compute for Apache Flink is earlier than V3.5.0, the following error message is displayed when you try to perform this operation: **An error occurred. System error: The BLINK version is abnormal. Error reason: blink version >= blink-3.5 is required, instance blink-3.4.4**.

To terminate a job, perform the following steps:

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration**.
3. On the **Administration** page, find the job that you want to terminate, and click **Terminate** in the **Actions** column.

> **Note** The **Terminate (checkpoint)** operation under **More** is different from the **Terminate** operation. The system triggers a checkpoint when you perform the **Terminate (checkpoint)** operation to terminate a job. Therefore, the time consumed to terminate a job by performing the **Terminate (checkpoint)** operation is longer than that by performing the **Terminate** operation. The job status is cleared after the job is terminated. The Terminate (checkpoint) operation has other functions in some scenarios. For example, if the upstream storage system is Message Queue for Apache Kafka, the system submits an offset each time it triggers a checkpoint. This ensures that the number of offsets submitted to the Kafka server is consistent with the amount of data consumed.

6.4. Job debugging

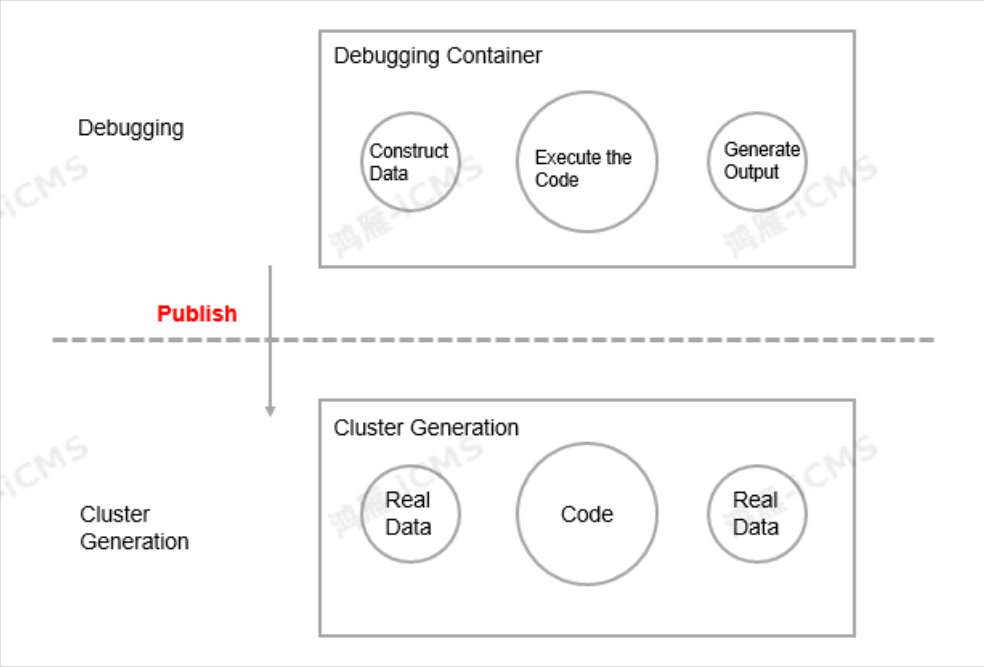
6.4.1. Perform local debugging

The Realtime Compute for Apache Flink development platform provides you with a local debugging environment. You can upload custom data, simulate job running, and check output in the local debugging environment to make sure that your business logic is valid.

Characteristics

The local debugging environment is completely isolated from the production environment. In the local debugging environment, all Flink SQL jobs run in an independent debugging container, and the debugging results are displayed on pages in the debugging environment. Local debugging does not affect online production streams, Realtime Compute for Apache Flink jobs, or data storage systems.

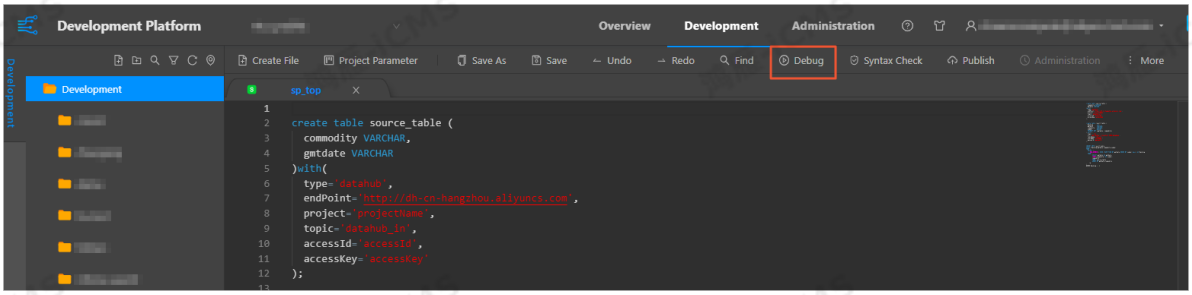
> **Note** In local debugging mode, Realtime Compute for Apache Flink cannot detect running failures caused by incompatible data formats in data storage resources. For example, Realtime Compute for Apache Flink cannot detect whether the length of output data exceeds the maximum length specified when you create an ApsaraDB RDS table.



Procedure

> **Note** Before you debug a job, make sure that you have developed the job. For more information, see Develop a job.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. On the top of the job edit section, click **Debug**.



4. In the **Debug File** dialog box, enter the test data. You can obtain test data by using one of the following methods:
   - Upload local data
     a. In the data preview section, click **Download Template**.
     b. Edit the debugging data based on the template.

     > **Note** The default delimiter for debugging data is a comma (,). For more information about how to define a delimiter, see Delimiter of the debugging data.

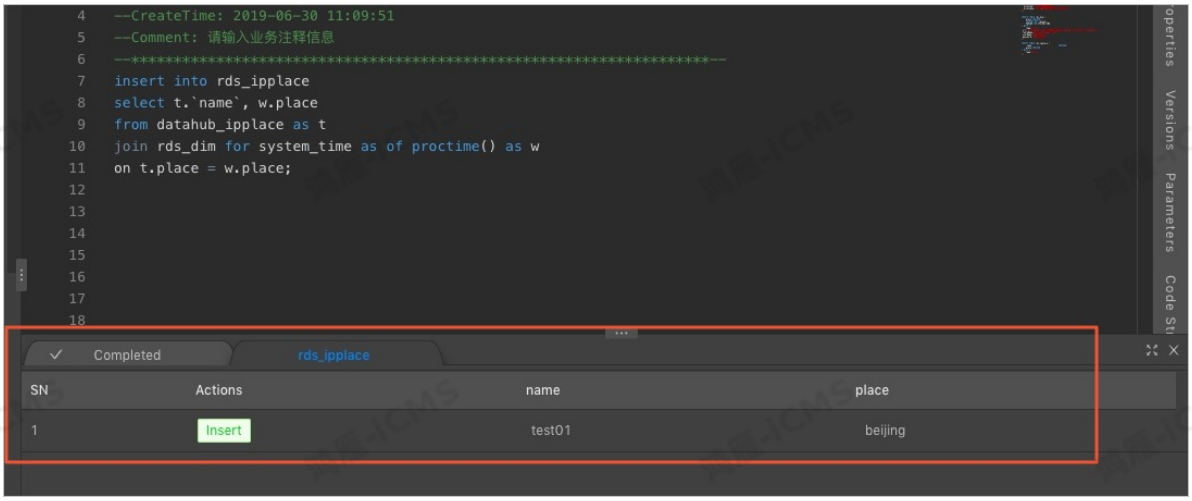
     c. In the **Upload File** section, click **Upload** to upload the debugging data.
   - Sample online data

     > **Note** Before you use the **Sequential Online Data Sampling** feature, make sure that the data source contains data during the sampling.

a. In the data preview section, click **Random Online Data Sampling** or **Sequential Online Data Sampling**.
b. Enter the sample configuration information.
c. Click **OK**.

> **Note** After the data is uploaded, you can view it in the data preview section.

5. Click **OK**.
6. In the lower part of the job edit section, view the debugging results.



Delimiter of the debugging data

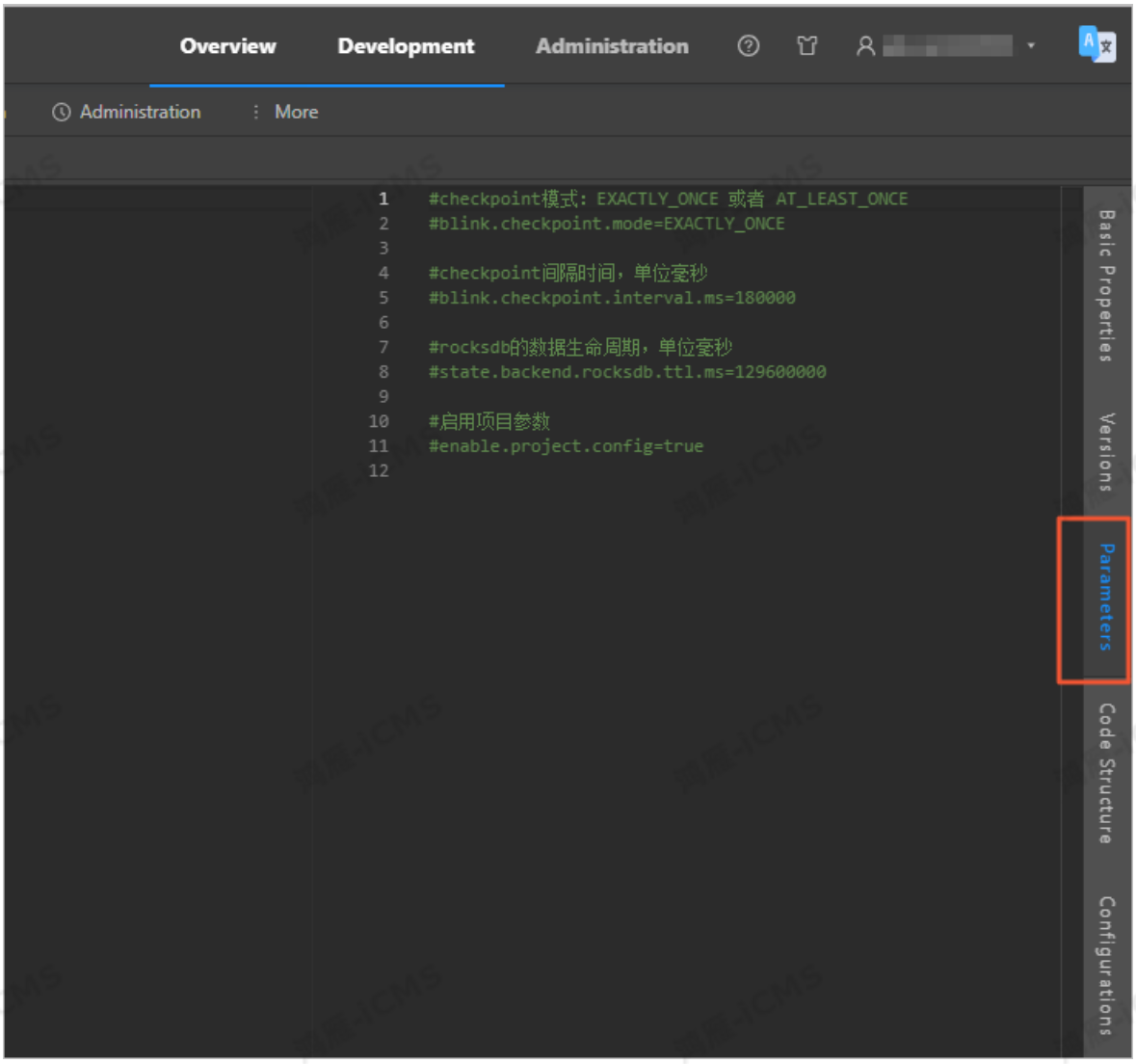
By default, the debugging data uses a comma (,) as the delimiter. If the input data, such as a JSON file, already contains commas (,), you must define another delimiter for the debugging data, such as a vertical bar (|).

> **Note** Realtime Compute for Apache Flink supports only a single character as a delimiter, such as a vertical bar (|). You cannot use a string as a delimiter, such as `aaa`.

To define a delimiter for debugging data, perform the following steps:

> **Note** Before you configure a delimiter, make sure that you have developed a job. For more information, see Develop a job.

1. On the right side of the job edit section, click **Parameters**.



2. In the parameter edit section of the job, enter the configuration of the delimiter. The following code shows how to configure a vertical bar (|) as the delimiter:

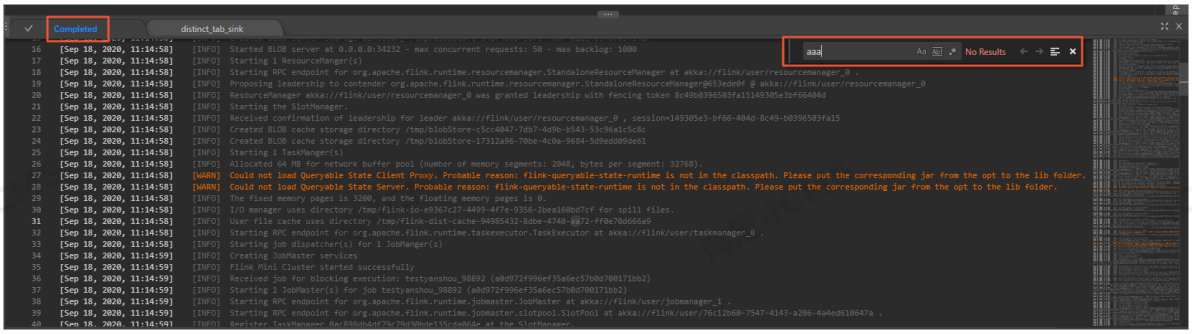
```
debug.input.delimiter = |
```

UDX logs

- Display user-defined extension (UDX) logs generated during local debugging.
  In Java, use the following method to convert the log format so the logs can be parsed by Realtime Compute for Apache Flink and display the UDX logs generated during local debugging:

```
public static void debugMsgOutput(String msg) {
    System.out.println(
      String.format("{\"type\":\"log\",\"level\": \"INFO\", \"time\": \"%s\",
\"message\": \"%s\", \"throwable\": \"null\"}\n",new SimpleDateFormat("yyyy-MM-dd HH:
mm:ss").format(new Date()), msg));
}
```

- View UDX logs.
  After the debugging is completed, you can view the UDX logs on the **Completed** tab at the lower part of the job edit section.



> **Note** You can press **Ctrl+F** to search for the logs.

6.4.2. Online debugging

The Realtime Compute for Apache Flink development platform provides an online debugging environment for you to debug your Realtime Compute for Apache Flink jobs. Compared with local debugging, online debugging consumes more compute units but validates business logic more accurately.

Online debugging uses real data storage resources to reduce the output differences between debugging and production. This helps you identify issues in the debugging phase.

Procedure

1. Develop a job. For more information, see Develop a job.
2. Modify the `type` parameter in the data definition language (DDL) statements of data storage resources.
   - Source table: `type = 'random'`
   - Result table: `type = 'print'`
3. Publish the job. For more information, see Publish a job.
4. Start the job. For more information, see Start a job.

Connectors

The Realtime Compute for Apache Flink development platform provides the following two types of connectors for online debugging:

- Source table `random`: periodically generates random data of a specific type.
- Result table `print`: generates computing results.

Parameters in a connector table

- random table

Parameter	Description
type	Required. The type of the connector. The value can only be random.
interval	Optional. The time interval at which data is generated. Unit: milliseconds. Default value: 500.

- print table

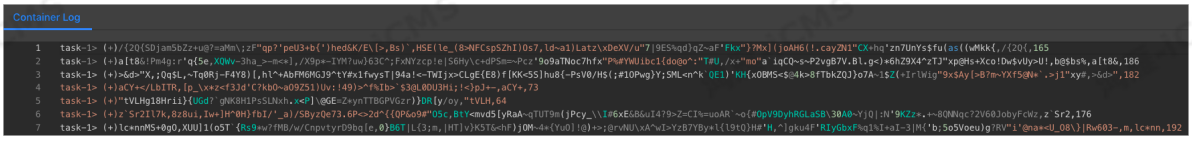
Parameter	Description
type	Required. The type of the connector. The value can only be print.
ignoreWrite	Optional. Specifies whether to ignore write operations. Default value: false. Valid values: ◦ false: Data is written to the result table, and logs are generated. ◦ true: Data is not written to the result table. The result table is empty, and no logs are generated.

Examples

- Test code

```
CREATE TABLE random_source (
  instr               VARCHAR
) WITH (
  type = 'random'
);

CREATE TABLE print_sink(
  instr VARCHAR,
  substr VARCHAR,
  num INT
)with(
  type = 'print'
);

INSERT INTO print_sink
SELECT
  instr,
  SUBSTRING(instr,0,5) AS substr,
  CHAR_LENGTH(instr) AS num
FROM random_source
```

- Test results



Query online debugging results

> **Note** Before you query the results of online debugging, make sure that you have published and started the job. For more information, see Publish a job and Start a job.

To query the results of online debugging, follow these steps:

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration** to go to the **Administration** page.
3. In the **Job Name** column, click the name of the job to go to the **Job Administration** page.

4. In the **Vertex Topology** section, click the required result table node.
5. On the top of the Execution Vertex page, navigate to **Subtask List > View Logs** to go to the **View Logs** page.
6. View the logs.
   - **Output of the print result table**

     Click **View Logs** in the Actions column of taskmanager.out.
   - **Output of UDX logs**
     If you use user defined extensions (UDXs), you can view the logs by using the following methods. For more information about how to use UDXs, see Overview.
     - **system.out or system.err method**

       Click **View Logs** in the Actions column of `taskmanager.out` or `taskmanager.err`.
     - **SLF4J Logger method**

       Click **View Logs** in the Actions column of `taskmanager.log`.

6.5. Job administration

6.5.1. Go to the Job Administration page

On the **Job Administration** page, you can view information about a job, such as the **running information**, **curve charts**, **failover information**, and **properties and parameters**. This topic describes how to go to the Job Administration page.

1. Log on to the Realtime Compute Console.
2. In the top navigation bar, click **Administration**.
3. In the **Jobs** section, click the target job name under the **Job Name** field.

6.5.2. Overview

The Overview page displays the real-time running information about a job. You can analyze and determine whether a job is healthy and meets your expectations based on the job status.

Go to the Overview page

1. Go to the **Administration** page in Realtime Compute for Apache Flink.
   i. Log on to the Realtime Compute development platform.
   ii. In the top navigation bar, click **Administration**.
   iii. On the **Jobs** page that appears, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Overview**.

Task status

The **Task Status** section displays the number of tasks in each state. A task can be in one of the following states:

- **Created**
- **Running**
- **Failed**
- **Completed**
- **Scheduling**
- **Canceling**

- **Canceled**

Job instantaneous values

Below the **Task Status** section, you can view the instantaneous values of a job.

Parameter	Description
Input TPS	The number of blocks that are read from the source table per second. For Log Service, you can package multiple data records into one log group for reading. In this case, the value indicates the number of log groups that are read from the source table per second.
Input RPS	The number of data records that are read from the source table per second.
Output RPS	The number of data records that are written to the result table per second.
Input BPS	The number of blocks that are read from the source table per second. Unit: bytes/s.
Consumed CUs	The current compute units (CUs) used by the job.
Start Time	The start time of the job.
Runtime	The duration for which the job has been running.

Vertex topology

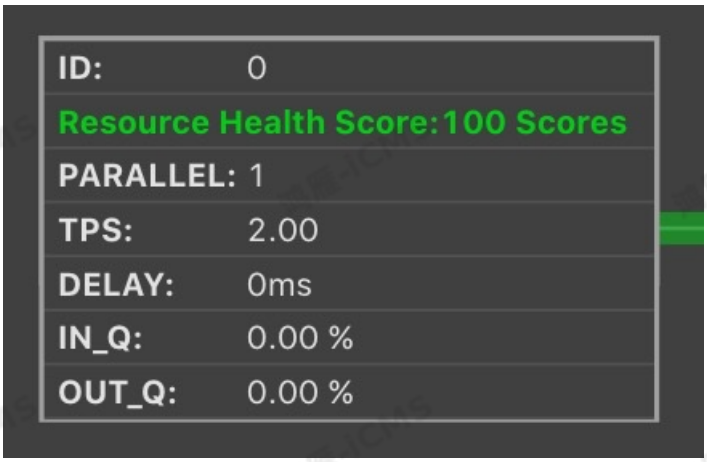
The **Vertex Topology** section displays the execution graph of the underlying computing logic of Realtime Compute for Apache Flink. Each component represents a task. Each data stream flows from one or more data source tables to one or more data result tables. The data flow resembles an arbitrary directed acyclic graph (DAG). To execute a job more efficiently, the underlying logic of Realtime Compute for Apache Flink chains subtasks to form an operator and chains operators to form a task. Each task is executed in a thread. This feature reduces thread switching, message serialization or deserialization, data swapping in the buffer, and data latency. However, this feature increases the overall throughput. An operator indicates the computational logic. A task is a collection of multiple operators.

- Display mode

  By default, the **Vertex Topology** section displays the topology. In the upper-right corner of the Vertex Topology section, you can click **List Mode** to change the display mode.

- Task status information

- Task parameters in view mode



In view mode, the information about the task is displayed in the related box. The following table describes the parameters.

Parameter	Description
Resource Health Score	The **resource health score** is obtained based on the specified check mechanism. The value indicates the job performance. If the **resource health score** is less than 60, data is stacked up for the task. This results in poor data processing performance. **Note** If the data processing performance is poor, we recommend that you optimize the performance by manual configuration. For more information, see Optimize performance by manual configuration.
PARALLEL	The parallelism for the task.
TPS	The number of blocks that are read from the source table per second.
DELAY	The processing delay of the task.
IN_Q	The percentage of the input queues for the task.
OUT_Q	The percentage of the output queues for the task.

- Task parameters in list mode
  At the bottom of the Vertex Topology section, you can view the information about the task in list mode. The following table describes the parameters of the task.

Parameter	Description
Name	The name of the task.
Status	The status of the task.
In Queue	The percentage of the input queues for the task.
Out Queue	The percentage of the output queues for the task.
Delay(ms)	The processing delay of the task.
TPS	The amount of data that is read from input nodes per second.
Bytes Received	The amount of data that is received by the task.
Records Received	The number of data records that are received by the task.
Bytes Sent	The amount of data that is sent from the task.
Records Sent	The number of data records that are sent from the task.
Task	The status of each parallelism for the task.

- Task thread information

  Click the task node. On the **SubTasks** tab, view the thread list of the task.

Vertex information

> **Note** The following features are supported only in Realtime Compute for Apache Flink V3.0.0 and later.

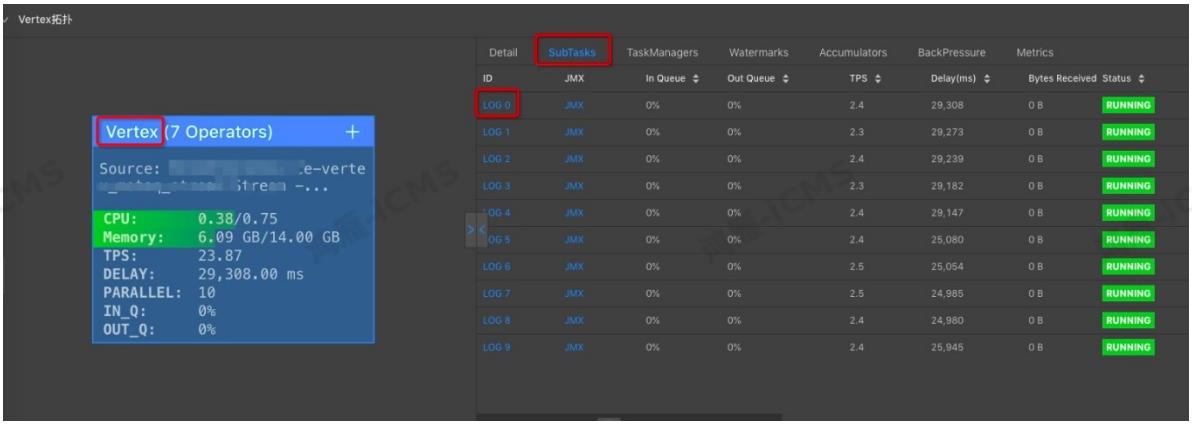
- Display Vertex operator information

  In the Vertex Topology section, click the plus sign (+) in the upper-right corner of a Vertex box to display the Vertex operator information.

- Display Vertex details

  In the upper-right corner of the Vertex Topology section, click **Expand All** to display Vertex details.

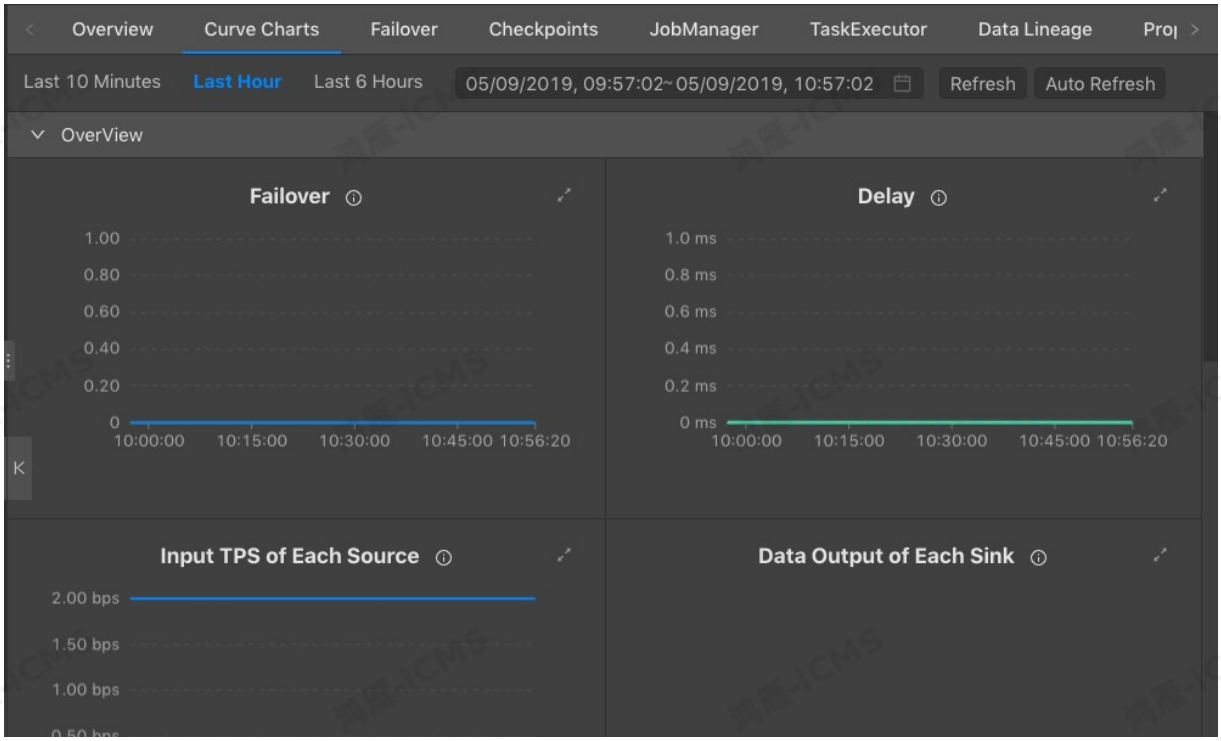
- Display the Vertex details page
  In the Vertex Topology section, click the Vertex border or the name in the Vertex list to display the Vertex details page on the right. On the Vertex details page, click the **SubTasks** tab. On the page that appears, click ID to go to the **TaskManager** page of the related log. For example, below the ID column, click **LOG 0**.



6.5.3. Metrics

Realtime Compute for Apache Flink shows core metrics of your job on the Curve Charts tab to help you diagnose the status of the job.

The following figure shows the curve chart of a metric.



> **Note**
>
> - The metrics are displayed only when a Realtime Compute for Apache Flink job is in the **running** state. If the job is in the **suspended** or **terminated** state, the metrics of this job are not displayed.
> - The metrics are collected and displayed on the Curve Charts tab after the job is running for more than one minute. This causes the latency in the data that is displayed in the curve charts.

Go to the Curve Charts tab

1. Go to the **Job Administration** page in the Realtime Compute for Apache Flink console.

i. Log on to the Realtime Compute development platform.
ii. In the top navigation bar, click **Administration**.
iii. On the **Jobs** page that appears, click the target job name under the **Job Name** field.

2. In the upper part of the **Job Administration** page, click the **Curve Charts** tab.

Overview

- **Failover**

  The failover curve chart displays the frequency of failovers that are caused by errors or exceptions for the current job. To calculate the failover rate, divide the total number of failovers that occurred within the minute that precedes the current failover time by 60. For example, if a failover occurred once within the last minute, the failover rate is 0.01667. The failover rate is calculated by using the following formula: **1/60 = 0.01667**.

- **Delay**
  To help you obtain the full-link timeliness and job performance, Realtime Compute for Apache Flink provides the following latency metrics:
  - **Processing Delay**: **Processing delay = Current system time - Event time at which the system processes the last data record**. If no more data enters upstream storage systems, the processing delay gradually increases as the system time continues to move forward.
  - **Data Pending Time**: **Data pending time = Time when data enters Realtime Compute for Apache Flink - Event time**. Even if no more data enters upstream storage systems, the queued time does not increase. The queued time is used to assess whether the Realtime Compute for Apache Flink job has backpressure.
  - **Data Arrival Interval**: **Data arrival interval = Processing delay - Data pending time**. If the Realtime Compute for Apache Flink job has no backpressure, the queued time is short and stable. In this case, this metric reflects the degree of data sparsity between the data sources. If the Realtime Compute for Apache Flink job has backpressure, the queued time is long or unstable. In this case, this metric cannot be used for reference.

  > **Note**
  > - Realtime Compute for Apache Flink uses a distributed computing framework. The preceding three latency metrics obtain values of each shard or partition of the data source. Then, the metrics report the maximum values among all the shards or all the partitions to the development platform of Realtime Compute for Apache Flink. Therefore, the aggregated data arrival interval that is displayed on the development platform is different from the interval that is obtained by using the following formula: **Data arrival interval = Processing delay - Data pending time**.
  > - If no more data enters a shard or a partition of the data source, the processing delay gradually increases.

- **Input TPS of Each Source**

  This chart displays statistics on all streaming data input of a Realtime Compute for Apache Flink job. The chart records the number of blocks that are read from the source table per second. This helps you obtain the transactions per second (TPS) of a data storage system. Different from TPS, the records per second (RPS) metric indicates the number of records read from the source table per second. These records are resolved from the blocks. For example, if Log Service reads five log groups per second, the value of TPS is 5. If eight log records are resolved from each log group, a total of 40 log records are resolved. In this case, the value of RPS is 40.

- **Data Output of Each Sink**

This chart displays statistics on all output data of a Realtime Compute for Apache Flink job. This helps you obtain the RPS of a data storage system. In most cases, if no data output is detected during system operations and maintenance (O&M), you must check the input of the upstream storage system and the output of the downstream storage system.

- **Input RPS of Each Source**

  This chart displays statistics on all input streaming data of a Realtime Compute for Apache Flink job. This helps you obtain the RPS of a data storage system. If no data output is detected during system O&M, you must check the RPS to determine whether the input data from the upstream storage system is normal.

- **Input BPS of Each Source**

  This chart displays statistics on all input streaming data of a Realtime Compute for Apache Flink job. This chart records the traffic that is used to read the input source table per second. This helps you obtain the bytes per second (BPS) of the traffic.

- **Dirty Data from Each Source**

  This chart displays the number of dirty data records in the data source of a Realtime Compute for Apache Flink job in different time periods.

- **Auto Scaling Successes and Failures**
  This chart displays the number of auto scaling successes and the number of auto scaling failures.

  > **Important** This curve chart is suitable only for Realtime Compute for Apache Flink whose version is later than V3.0.0.

- **CPUs Consumed By Auto Scaling**
  This chart displays the number of CPUs consumed when auto scaling is performed.

  > **Important** This curve chart is suitable only for Realtime Compute for Apache Flink whose version is later than V3.0.0.

- **Memory Consumed by Auto Scaling**
  This chart displays the memory space consumed when auto scaling is performed.

  > **Important** This curve chart is suitable only for Realtime Compute for Apache Flink whose version is later than V3.0.0.

Advanced View

Alibaba Cloud Realtime Compute for Apache Flink provides a fault tolerance mechanism that allows you to restore data streams and ensures that the data streams are consistent with the application. The fault tolerance mechanism is used to create consistent snapshots of distributed data streams and the related states. These snapshots work as consistency checkpoints to which the system can fall back if a failure occurs.

One of the core concepts for distributed snapshots is barriers. Barriers are inserted into data streams and flow together with the data streams to the downstream. Barriers do not overtake data records. The records flow strictly in line. A barrier divides a data stream into two parts. One part enters the current snapshot and the other part enters the next snapshot. Each barrier has a snapshot ID. If the data flows before a barrier is inserted in the data stream, the data is included in the specified snapshot. Barriers are lightweight. Barriers do not interfere with the processing of data streams. Multiple barriers from different snapshots can co-exist in the same data stream. This allows multiple snapshots to be concurrently created.
Barriers are inserted into data streams at the data source. If a barrier from Snapshot n is inserted, the system automatically records the checkpoint of Snapshot n in the data stream. This checkpoint is indicated by Sn. Then, the barrier continues to flow to the downstream.

When an intermediate operator receives barriers for Snapshot n from all the input streams, the operator emits only one barrier for Snapshot n to all the output streams. When the sink operator that is the destination of the directed acyclic graph (DAG) stream receives Barrier n from each of its input streams, the operator acknowledges to the checkpoint coordinator that Snapshot n is created. After all the sink operators acknowledge that Snapshot n is created, this snapshot is considered completed. The following table describes the curve charts of checkpoint metrics.

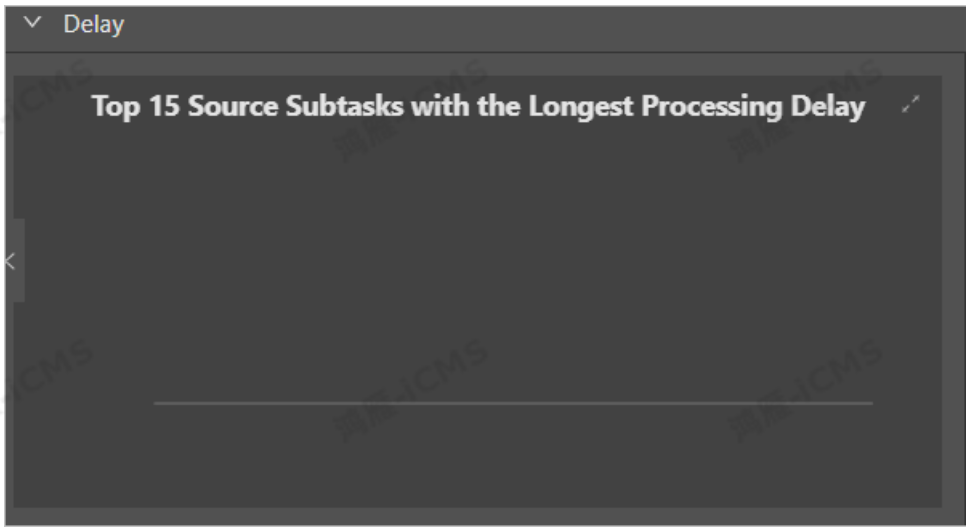
Curve chart	Description
Checkpoint Duration	Displays the time that is consumed to create a checkpoint. Unit: milliseconds.
Checkpoint Size	Displays the memory size that is required to create a checkpoint.
Checkpoint Alignment Time	Displays the duration consumed by all the data streams to flow from the upstream nodes to the node on which you create a checkpoint. When the sink operator receives Barrier n from all the input streams, the operator acknowledges to the checkpoint coordinator that Snapshot n is created. The sink operator represents the destination of the DAG stream. After all the sink operators acknowledge that snapshot n is created, this snapshot is considered completed. This duration is known as the checkpoint alignment time.
Checkpoint Count	Displays the number of checkpoints within a specific period of time.
Get	Displays the longest duration for which a subtask performs a GET operation on the RocksDB within a specific period of time.
Put	Displays the longest duration for which a subtask performs a PUT operation on the RocksDB within a specific period of time.
Seek	Displays the longest duration for which a subtask performs a SEEK operation on the RocksDB within a specific period of time.
State Size	Displays the state size of the job within a specific period of time. If the size increases at a high rate, we recommend that you check for potential issues in the job.
GMS GC Time	Displays the duration for which the underlying container of the job performs garbage collection.
GMS GC Rate	Displays the frequency at which the underlying container of the job performs garbage collection.

Watermark

Curve chart	Description
Watermark Delay	Displays the difference between the watermark time and the system time.
Dropped Records per Second	Displays the number of data records that are dropped per second. If a data record arrives at the window after the watermark time, the data record is dropped.
Dropped Records	Displays the total number of dropped data records. If a data record arrives at the window after the watermark time, the data record is dropped.

Delay

Top 15 Source Subtasks with the Longest Processing Delay



This chart displays the processing delay of each source subtask.

Throughput

Curve chart	Description
Task Input TPS	Displays the data input status of all the tasks in a job.
Task Output TPS	Displays the data output status of all the tasks in a job.

Queue

Curve chart	Description
Input Queue Usage	Displays the data input queue of all the tasks in a job.
Output Queue Usage	Displays the data output queue of all the tasks in a job.

Tracing

Curve chart	Description
Time Used In Processing Per Second	Displays the duration for which a task processes data per second.
Time Used In Waiting Output Per Second	Displays the duration for which a task waits for output data per second.
Task Latency Histogram Mean	Displays the latency of each task.
Wait Output Histogram Mean	Displays the duration for which each task waits for output.
Wait Input Histogram Mean	Displays the duration for which each task waits for input.
Partition Latency Mean	Displays the latency of concurrent tasks in each partition.

Process

Curve chart	Description
Process Memory RSS	Displays the memory usage of each process.
CPU Usage	Displays the CPU utilization of each process.

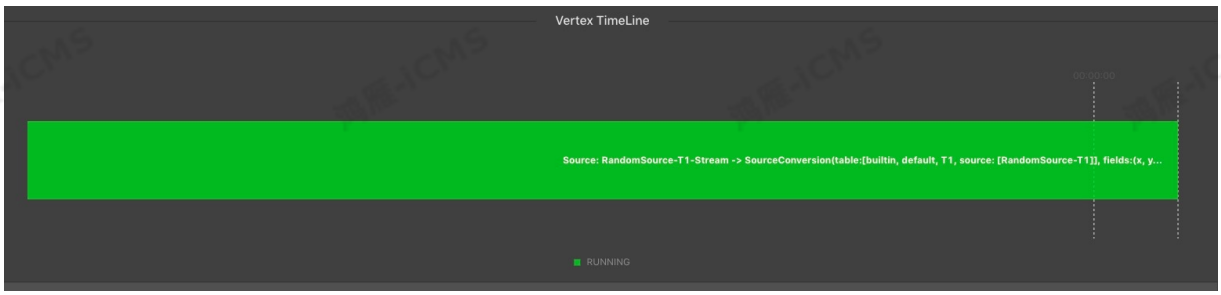
JVM

Curve chart	Description
Memory Heap Used	Displays the Java Virtual Machine (JVM) heap memory usage of a job.
Memory Non-Heap Used	Displays the JVM non-heap memory usage of a job.
Thread Count	Displays the number of threads in a job.
GC(CMS)	Displays the number of times that a job completes garbage collection (GC).

6.5.4. Timeline

The Timeline page displays the running status of each Vertex from the start offset to the current time.

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.



Go to the Timeline page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Timeline**.

6.5.5. Failover

Alibaba Cloud Realtime Compute provides the Failover page for the current job. On the Failover page, you can view the running status and error messages of the current job.

Go to the Failover page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Failover**.

Latest FailOver

The **Latest FailOver** tab displays the current errors of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

FailOver History

The **FailOver History** tab displays the historical errors of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

Root Exception

The **Root Exception** tab displays the current exceptions of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

Exception History

The **Exception History** tab displays the historical exceptions of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

6.5.6. Checkpoints

Alibaba Cloud Realtime Compute provides a fault tolerance that allows you to restore data streams and make sure that the data streams are consistent with the application. The central part of the fault tolerance is to create consistent snapshots of distributed data streams and their states. These snapshots act as consistency checkpoints to which the system can fall back when a failure occurs.

Go to the Checkpoints page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Checkpoints**.

Overview

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **Overview** tab displays the latest checkpoint information, such as the process, duration, and state size of the checkpoint at each node.

History

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **History** tab displays the recent checkpoint information. Click the plus sign (+) at the beginning of the row to display the checkpoint information, such as the process, duration, and state size of the checkpoint at each node.

Summary

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **Summary** tab displays the average, maximum, and minimum values of completed checkpoints.

Configuration

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **Configuration** tab displays the configuration information of the checkpoints.

Completed Checkpoints

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

The **Completed Checkpoints** tab displays the information about the completed checkpoints.

Parameter	Description
ID	The ID of the checkpoint.
Start Time	The start time of the checkpoint.
Durations (ms)	The time spent on creating the checkpoint.

Task Latest Completed Checkpoint

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

The **Task Latest Completed Checkpoint** tab displays the details about the latest checkpoint.

Parameter	Description
SubTask ID	The ID of the subtask.
State Size (Bytes)	The state size of the checkpoint.
Durations (ms)	The time spent on creating the checkpoint.

6.5.7. JobManager

JobManager plays an important part in the startup process of a Realtime Compute for Apache Flink cluster. You can view the JobManager parameter information on the JobManager tab.

Go to the JobManager tab

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute development platform.
   ii. In the top navigation bar, click **Administration**.
   iii. On the **Jobs** page that appears, click the target job name under the **Job Name** field.
2. On the **Job Administration** page, click the **JobManager** tab.

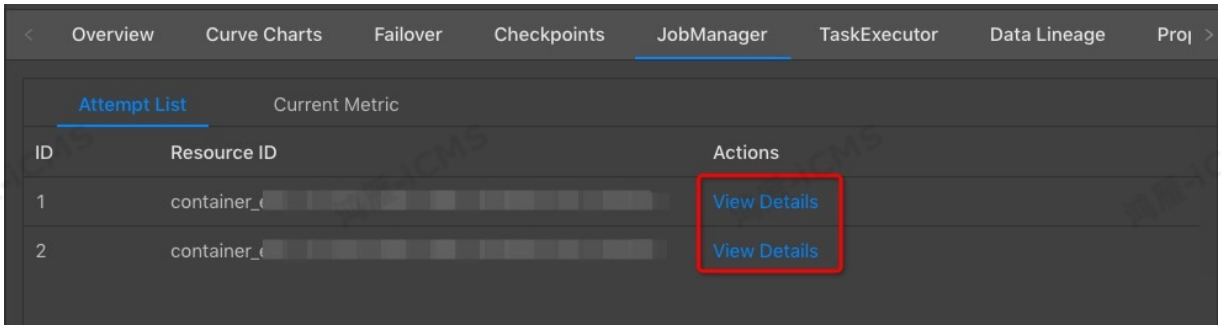
Role of JobManager in the cluster startup process

JobManager plays an important part in the startup process of a Realtime Compute for Apache Flink cluster. The following items describe the startup process of a Realtime Compute for Apache Flink cluster:

1. When a Realtime Compute for Apache Flink cluster is started, one JobManager and several TaskExecutors are started at the same time.
2. The client submits tasks to the JobManager.
3. The JobManager assigns tasks to TaskExecutors.
4. The TaskExecutors report the heartbeat and statistical information to the JobManager.

JobManager parameters

On the Job Administration page, click the **JobManager** tab. On the **Attempt List** tab, click **View Details** in the **Actions** column to view detailed information about the JobManager.



6.5.8. TaskExecutor

This topic describes the role of TaskExecutors in the startup process of a Realtime Compute for Apache Flink cluster and explains the TaskExecutor tab.

> **Important** This topic applies to only Realtime Compute for Apache Flink whose version is earlier than V3.0.

Background information

TaskExecutors are an indispensable part in starting a Realtime Compute for Apache Flink cluster. TaskExecutors receive tasks from and return execution results to the JobManager. The number of slots is specified when a TaskExecutor is started. Only one task thread can be executed in each slot. A TaskExecutor receives tasks from the JobManager, and then builds a Netty connection with its upstream to receive and process data.

Go to the TaskExecutor tab

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration**.
3. In the **Jobs** section, click the name of the required job under the **Job Name** field.
4. On the **Job Administration** page, click the **TaskExecutor** tab.

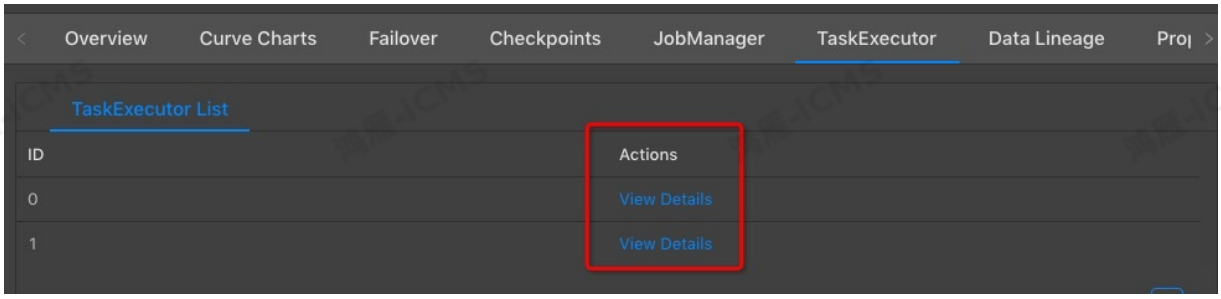
Role of TaskExecutors in cluster startup

TaskExecutors are an indispensable part in starting a Realtime Compute for Apache Flink cluster. The following items describe the startup process of a Realtime Compute for Apache Flink cluster:

1. When a Realtime Compute for Apache Flink cluster is started, one JobManager and several TaskExecutors are started at the same time.
2. The client submits tasks to the JobManager.
3. The JobManager assigns tasks to TaskExecutors.
4. The TaskExecutors report the heartbeat and statistical information to the JobManager.

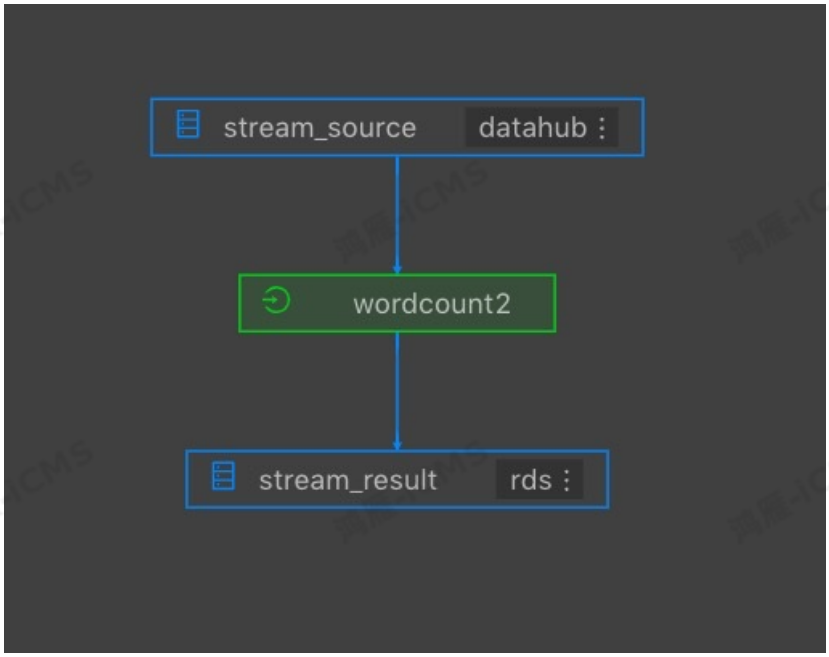
TaskExecutor tab

The TaskExecutor tab provides a list of tasks and the entries to their details.



6.5.9. Data lineage

The data lineage of a Realtime Compute job reflects the dependency between upstream and downstream data of the job. In scenarios where the business dependency between the upstream and downstream data of a job is complex, Realtime Compute provides a data topology on the Data Lineage page to clearly show the dependency.



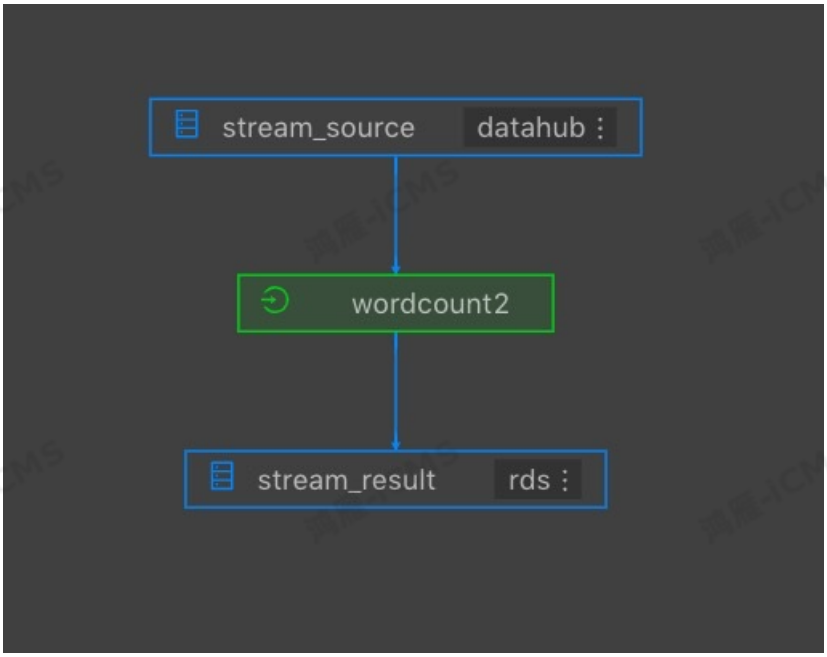
Go to the Data Lineage page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Data Lineage**.

Data sampling

The Data Lineage page provides the data sampling feature for source tables and result tables of jobs. The data to be sampled is the same as the data displayed on the data development page. The data sampling feature allows you to check data at any time on the data administration page, thus facilitating fault locating. To enable the data sampling feature, follow these steps:

1. Click the table name in the upstream and downstream of the job.



2. At the bottom of the **Data Sampling** page, click **OK**.

6.5.10. Properties and parameters

The Properties and Parameters tab provides detailed information about the current job, such as the current running information and running history.

Go to the Properties and Parameters tab

1. Go to the **Job Administration** page in the Realtime Compute for Apache Flink console.
   i. Log on to the Realtime Compute development platform.
   ii. In the top navigation bar, click **Administration**.
   iii. On the **Jobs** page that appears, click the target job name under the **Job Name** field.
2. On the **Job Administration** page, click the **Properties and Parameters** tab.

Code

On the **Code** tab, you can preview the SQL job code. In the upper-right corner of the Code tab, click **Edit Job** to go to the **Development** page.

Resource Configuration

- On the **Resource Configuration** tab, you can view the configuration of the resources that are used in a job, such as CPUs, memory, and parallelism.
- After auto scaling is enabled, you can query the auto scaling iteration history on the Resource Configuration tab.

> **Note** Only Realtime Compute for Apache Flink V3.0.0 and later allow you to query the auto scaling iteration history.

Properties

On the **Properties** tab, you can view basic information about a job.

Runtime Parameters

On the **Runtime Parameters** tab, you can view the job running parameters, such as the underlying checkpoint and start time.

History

On the **History** tab, you can view the operation information about a job, such as **Operated By**, **Start Offset**, and **End Time**.

Parameters

On the **Parameters** tab, you can specify the job parameters supported by Realtime Compute for Apache Flink. For example, you can customize the delimiter for debugging.

6.5.11. Job diagnosis

Realtime Compute offers job diagnosis to help you troubleshoot job issues.

Procedure

> **Note** Only jobs that are in the running state can be diagnosed.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration**.
3. On the Administration page that appears, find the target job, move the pointer over the More icon in the **Actions** column, and click **Check**.

Check metrics

- **Failover**
  - **Job failover**: Check whether the job encountered a failover within the last 30 minutes.
  - **Application Master (AM) failover**: Check whether a failover is detected in AM.
- **Blink Metric Job latency**: Check the job latency. If a latency occurs, the nodes with backpressure are displayed.
  - **High latency**: The latency is longer than 100 seconds and shorter than 200 seconds.
  - **Excessively high latency**: The latency is 200 seconds or longer.
- This tab displays the Yarn check result.
- This tab displays the **OS** check result.

6.6. Job optimization

6.6.1. Overview

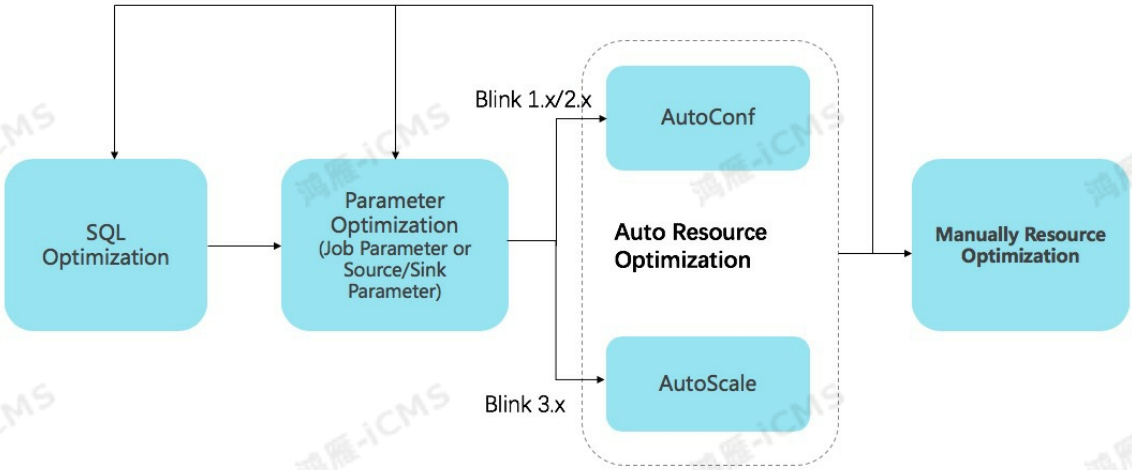
After you implement the business logic, and then publish and start a Realtime Compute job, you need to optimize the job to meet performance requirements.

Purposes

- Jobs can start and run properly.
- The job latency and throughput meet performance requirements.
- Resources can be used efficiently to reduce the cost.

Procedure

The following figure shows the procedure of job optimization.



1. Optimize the SQL code.

   SQL optimization allows you to select an appropriate SQL implementation method based on business requirements. For example, you can optimize aggregation functions, resolve data hotspot issues, optimize the TopN algorithm, use built-in functions, deduplicate data records, and avoid use of regular expressions. For more information, see Recommended Flink SQL practices.

2. Optimize performance by adjusting parameter settings.
   - Adjust job parameter settings.

     Select an underlying optimization policy. For example, you can enable miniBatch to reduce state data access. For more information, see Job parameters.
   - Adjust parameter settings of upstream and downstream data storage.

     Optimize the read and write operations performed on the upstream and downstream storage systems. For example, you can read or write data in batches to improve the throughput. You can also configure the cache policy to improve the efficiency of joining dimension tables. For more information, see Upstream and downstream storage parameters.

3. Optimize resource configuration automatically.

   To simplify job optimization, Realtime Compute provides the automatic configuration optimization feature. We recommend that you use this feature for job optimization. For more information, see Performance optimization by using auto scaling.

4. Optimize resource configuration manually or repeat the optimization process.
   - Optimize resource configuration manually.
     If automatic configuration optimization cannot meet your requirements, you can manually optimize the resource configuration. For more information, see Optimize performance by manual configuration.
   - Repeat the optimization process.

     If the optimization result cannot meet your requirements, repeat the previous steps.

6.6.2. Recommended Flink SQL practices

This topic describes the recommended syntax, configurations, and functions used to optimize Flink SQL performance.

Optimize the Group By functions

- Enable microBatch or miniBatch to improve the throughput

  The microBatch and miniBatch policies are both used for micro-batch processing. If either of the policies is enabled, Realtime Compute for Apache Flink processes data when the data cache meets the trigger condition. This reduces the frequency at which Realtime Compute for Apache Flink accesses the state data, and therefore improves the throughput and reduces data output.

  The microBatch and miniBatch policies are different from each other in terms of the trigger mechanism. The miniBatch policy triggers micro-batch processing by using the timer threads that are registered with each task. This consumes some thread scheduling overheads. The microBatch policy is an enhancement of the miniBatch policy. The microBatch policy triggers micro-batch processing based on event messages, which are inserted into the data sources at a specific interval. The microBatch policy outperforms the miniBatch policy because it provides higher data serialization efficiency, reduces backpressure, and achieves higher throughput at a lower latency.

  - Use scenarios
    Micro-batch processing achieves higher throughput at the expense of higher latency. We recommend that you do not enable micro-batch processing in scenarios that require extremely low latency. However, in data aggregation scenarios, we recommend that you enable micro-batch processing to improve job performance.

    > **Note** You can also enable microBatch to resolve data jitter when data is aggregated in two phases.

  - Enabling method
    microBatch and miniBatch are disabled by default. To enable them, configure the following parameters:

    ```
    # Enable window miniBatch in Realtime Compute for Apache Flink V3.2 or later. By default, window miniBatch is disabled for Realtime Compute for Apache Flink V3.2 or later.
    sql.exec.mini-batch.window.enabled=true
    # The interval at which a large amount of data is generated. You must specify this parameter when you enable microBatch. We recommend that you set this parameter to the same value as that of blink.miniBatch.allowLatencyMs.
    blink.microBatch.allowLatencyMs=5000
    # When you enable microBatch, you must reserve the settings of the following two miniBatch parameters:
    blink.miniBatch.allowLatencyMs=5000
    # The maximum number of data records that can be cached for each batch. You must set this parameter to avoid the out of memory (OOM) error.
    blink.miniBatch.size=20000
    ```

- Enable LocalGlobal to resolve common data hotspot issues

  The LocalGlobal policy divides the aggregation process into two phases: local aggregation and global aggregation. They are similar to the combine and reduce phases in MapReduce. In the local aggregation phase, Realtime Compute for Apache Flink locally aggregates a micro batch of data at each input node (LocalAgg), and generates an accumulator value for each batch (accumulator). In the global aggregation phase, Realtime Compute for Apache Flink merges the accumulator values (merge) to obtain the final result (GlobalAgg).

  The LocalGlobal policy can eliminate data skew by using local aggregation and resolve data hotspot issues in global aggregation. Therefore, job performance is enhanced.

- Use scenarios
  You can enable LocalGlobal to improve the performance of general aggregate functions, such as SUM, COUNT, MAX, MIN, and AVG, and resolve data hotspot issues when you execute these functions.

  > **Note** To enable LocalGlobal, you must define a user-defined aggregate function (UDAF) to implement the `merge` method.

- Enabling method

  In Realtime Compute for Apache Flink V2.0 or later, LocalGlobal is enabled by default. When the **blink.localAgg.enabled** parameter is set to true, LocalGlobal is enabled. This parameter takes effect only when **microBatch** or **miniBatch** is enabled.

- Verification

  To determine whether LocalGlobal is enabled, check whether the **GlobalGroupAggregate** or **LocalGroupAggregate** node exists in the generated topology.

- Enable PartialFinal to resolve data hotspot issues when you execute the COUNT DISTINCT function

  The LocalGlobal policy effectively improves the performance of general aggregate functions, such as SUM, COUNT, MAX, MIN, and AVG. However, it is not effective for improving the performance of the COUNT DISTINCT function. This is because local aggregation cannot effectively remove duplicate distinct keys. As a result, a large amount of data remains stacked up in the global aggregation phase.

  If you execute the COUNT DISTINCT function in Realtime Compute for Apache Flink versions earlier than V2.2.0, you must add a layer that scatters data by a distinct key so that you can divide the aggregation process into two phases to resolve data hotspot issues. Realtime Compute for Apache Flink `V2.2.0` and later versions provide the PartialFinal policy to automatically scatter data and divide the aggregation process.

  - Use scenarios
    The PartialFinal policy applies to scenarios in which the aggregation performance cannot meet your requirements when you use the COUNT DISTINCT function.

    > **Note**
    > - You cannot enable PartialFinal in the Flink SQL code that contains UDAFs.
    > - We recommend that you enable PartialFinal only when the amount of data is large. This is because the PartialFinal policy automatically scatters data to two aggregation layers and introduces additional network shuffling. If the amount of data is not large, resources are wasted.

  - Enabling method

    PartialFinal is disabled by default. To enable PartialFinal, set the `blink.partialAgg.enabled` parameter to true.

  - Verification

    To determine whether PartialFinal is enabled, check whether **expandable** nodes exist in the generated topology, or whether the number of aggregation layers changes from one to two.

- Use the AGG WITH FILTER syntax to improve job performance when you use the COUNT DISTINCT function

> **Note** Only Realtime Compute for Apache Flink V2.2.2 and later versions support this syntax.

Statistical jobs record unique visitors (UVs) in different dimensions, such as UVs of the entire network, UVs of mobile clients, and UVs of PCs. We recommend that you use the standard AGG WITH FILTER syntax instead of AGG WITH CASE WHEN to implement multi-dimensional statistical analysis. The SQL optimizer of Realtime Compute for Apache Flink can analyze the filter parameter. This way, Realtime Compute for Apache Flink can execute the COUNT DISTINCT function on the same field with different filter conditions by sharing the state data. This reduces the read and write operations on the state data. The performance test shows that the use of AGG WITH FILTER improves job performance by one time higher than the use of AGG WITH CASE WHEN.

- Use scenarios

  We recommend that you replace the AGG WITH CASE WHEN syntax with the AGG WITH FILTER syntax. This particularly improves job performance when you execute the COUNT DISTINCT function on the same field with different filter conditions.

- Original statement

```
COUNT(distinct visitor_id) as UV1 , COUNT(distinct case when is_wireless='y' then visitor_id else null end) as UV2
```

- Optimized statement

```
COUNT(distinct visitor_id) as UV1 , COUNT(distinct visitor_id) filter (where is_wireless='y') as UV2
```

Optimize the TopN algorithm

- TopN algorithm
  If the input streams of TopN are static streams (such as source), TopN supports only one algorithm: AppendRank. If the input streams of TopN are dynamic streams (such as streams that are processed by using the AGG or JOIN function), TopN supports the following three algorithms in descending order of performance: UpdateFastRank, UnaryUpdateRank, and RetractRank. The name of the algorithm used is contained in the node name in the topology.
  - UpdateFastRank is the optimal algorithm.
    The following two conditions must be met if you want to use this algorithm:
    - The input streams must contain the primary key information, such as ORDER BY AVG.
    - The values of the fields or functions in the ORDER BY clause are updated monotonically in the opposite order of sorting. For example, you can define the ORDER BY clause as ORDER BY COUNT, ORDER BY COUNT_DISTINCT, or ORDER BY SUM (positive) DESC. This optimization is supported only in Realtime Compute for Apache Flink V2.2.2 or later.
      If you want to obtain an optimization plan, you must add a filter condition in which the SUM parameter is positive when you use ORDER BY SUM DESC, and make sure that the value of **total_fee** is positive.

```
insert
  into print_test
SELECT
  cate_id,
  seller_id,
  stat_date,
  pay_ord_amt -- The rownum field is not included in the output data. This reduces the amount of output data to be written to the result table.
FROM (
    SELECT
      *,
      ROW_NUMBER () OVER (
        PARTITION BY cate_id,
        stat_date -- Ensure that the stat_date field is included. Otherwise, the data may be disordered when the state data expires.
        ORDER
          BY pay_ord_amt DESC
      ) as rownum -- Sort data by the sum of the input data.
    FROM (
        SELECT
          cate_id,
          seller_id,
          stat_date,
          -- Note: The result of the SUM function is monotonically increasing because the values returned by the SUM function are positive. Therefore. TopN can use optimized algorithms to obtain top 100 data records.
          sum (total_fee) filter (
            where
              total_fee >= 0
          ) as pay_ord_amt
        FROM
          random_test
        WHERE
          total_fee >= 0
        GROUP
          BY cate_name,
          seller_id,
          stat_date
      ) a
  )
WHERE rownum <= 100;
```

    - UnaryUpdateRank is second only to UpdateFastRank in terms of performance. To use this algorithm, make sure that the input streams contain the primary key information.
    - RetractRank ranks last in terms of performance. We recommend that you do not use this algorithm in the production environment. Check input streams. If input streams contain the primary key information, use UnaryUpdateRank or UpdateFastRank to optimize job performance.
- Optimization method
    - Exclude the rownum field

Do not include rownum in the output of TopN. We recommend that you sort the results immediately after they are displayed in the frontend. This can significantly reduce the amount of data that needs to be written to the result table. For more information, see TopN.

- Increase the cache size of TopN
  TopN provides a state cache to improve the access efficiency of state data. This improves the performance. The following formula is used to calculate the hit rate of TopN cache:

  ```
  cache_hit = cache_size*parallelism/top_n/partition_key_num
  ```

  Take Top100 as an example. Assume that the cache contains 10,000 records and the parallelism is 50. If the number of keys for the PARTITION BY function is 100,000, the cache hit rate equals 5% (10000 × 50/100/100000 = 5%). The hit rate is low, which indicates that large amounts of requests will access the disk state data. As a result, job performance significantly deteriorates. Therefore, if the number of keys for the PARTITION BY function is large, you may increase the cache size and heap memory of TopN. For more information, see Optimize performance by manual configuration.

  ```
  ## In this example, if you increase the cache size of TopN from the default value 10000 to 200000, the cache hit rate may reach 100% (200000 × 50/100/100000 = 100%).
  blink.topn.cache.size=200000
  ```

- Include a time field in the PARTITION BY function

  For example, you want to include the day field in your statement for a daily ranking. Otherwise, the TopN result may become disordered when the state data expires.

Optimize the deduplication performance

> **Note** Only Blink 3.2.1 supports efficient deduplication solutions.

Input streams of Realtime Compute for Apache Flink may contain duplicate data. Therefore, deduplication is highly required. Realtime Compute for Apache Flink offers two policies to efficiently remove duplicate data: Deduplicate Keep FirstRow and Deduplicate Keep LastRow.

- Syntax
  Flink SQL does not support deduplication statements. To reserve the first or last duplicate record under the specified primary key and discard the rest of the duplicate records as required, Realtime Compute for Apache Flink uses the ROW_NUMBER OVER WINDOW statement of Flink SQL. Deduplication is a special TopN statement.

  ```
  SELECT *
  FROM (
     SELECT *,
      ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]
        ORDER BY timeAttributeCol [asc|desc]) AS rownum
     FROM table_name)
  WHERE rownum = 1
  ```

Element	Description
ROW_NUMBER()	Specifies an over window to compute the row number. The row number starts from 1.
PARTITION BY col1[, col2..]	Optional. Specifies the partition key columns that store primary keys of duplicate records.

ORDER BY timeAttributeCol [asc|desc])	Specifies the column that sorts records based on a time attribute (proctime or rowtime). You can sort records in ascending order (Deduplicate Keep FirstRow) or descending order (Deduplicate Keep LastRow) based on the time attribute.
rownum	Specifies the number of rows. You can set this element in either of the following ways: `rownum = 1` and `rownum <= 1`.

Based on the preceding syntax, deduplication includes two steps:

i. Use the `ROW_NUMBER()` window function to sort data by the specified time attribute and mark the data with rankings.

> **Note**
> - If the time attribute is proctime, Realtime Compute for Apache Flink removes duplicate records based on the time at which the records are processed by Realtime Compute for Apache Flink. In this case, the ranking may vary each time.
> - If the time attribute is rowtime, Realtime Compute for Apache Flink removes duplicate records based on the time at which the records are written to Realtime Compute for Apache Flink. In this case, the ranking always remains the same.

ii. Reserve the first record under the specified primary key and remove the rest of the duplicate records.

> **Note** You can sort records in ascending or descending order of the time attribute.
> - Deduplicate Keep FirstRow: Realtime Compute for Apache Flink sorts records in ascending order of the time attribute and reserves the first record under the specified primary key.
> - Deduplicate Keep LastRow: Realtime Compute for Apache Flink sorts records in descending order of the time attribute and reserves the first record under the specified primary key.

- Deduplicate Keep FirstRow
  If you select the Deduplicate Keep FirstRow policy, Realtime Compute for Apache Flink reserves the first record under the specified primary key but discards the rest of the duplicate records. In this case, the state data stores only the primary key information, and the access efficiency of the state data is significantly improved. The following sample code is an example:

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b ORDER BY proctime) as rowNum
  FROM T
)
WHERE rowNum = 1
```

> **Note** The preceding code removes duplicate records from table T based on the b field, and reserves the first record under the specified primary key based on the system time. The proctime attribute indicates the processing time attribute. Realtime Compute for Apache Flink sorts data records in table T based on this attribute. To remove duplicate records based on the system time, you can also call the `PROCTIME` function to avoid the need to declare the proctime attribute.

- Deduplicate Keep LastRow
  If you select the Deduplicate Keep LastRow policy, Realtime Compute for Apache Flink reserves the last record under the specified primary key and discards the rest of the duplicate records. This policy slightly outperforms the LAST_VALUE function in terms of performance. The following sample code of Deduplicate Keep LastRow is an example:

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY rowtime DESC) as rowNum
  FROM T
)
WHERE rowNum = 1
```

> **Note** The preceding code removes duplicate records in table T based on the b and d fields, and reserves the last record under the specified primary key based on the time at which the records are written to Realtime Compute for Apache Flink. The rowtime attribute indicates the event time at which the records are written to Realtime Compute for Apache Flink. Realtime Compute for Apache Flink sorts records in table T based on this attribute.

Use efficient built-in functions

- Use built-in functions to replace user-defined extensions (UDXs)
  Built-in functions of Realtime Compute for Apache Flink are under continual optimization. We recommend that you use built-in functions to replace UDXs whenever possible. Realtime Compute for Apache Flink V2.0 optimizes built-in functions in the following aspects:
  - Improves serialization and deserialization efficiency.
  - Allows operations at the byte level.
- Use single-character delimiters in the KEYVALUE function

  The signature of the KEYVALUE function is `KEYVALUE(content, keyValueSplit, keySplit, keyName)`. When keyValueSplit and KeySplit are single-character delimiters, such as a colon (:) or a comma (,), Realtime Compute for Apache Flink uses an optimization algorithm. Realtime Compute for Apache Flink directly searches for the required keyName values among the binary data without the need to segment the entire content. This improves job performance by approximately 30%.
- Use the MULTI_KEYVALUE function when multiple key-value pairs exist

  > **Note** The MULTI_KEYVALUE function is supported only in Realtime Compute for Apache Flink V2.2.2 or later.

  Job performance is significantly affected if a query involves multiple KEYVALUE functions on the same content. Assume that the content contains 10 key-value pairs. To extract all the 10 values and use them as fields, you must write 10 KEYVALUE functions to parse the content 10 times. As a result, job performance deteriorates.

In this case, we recommend that you use the table-valued function MULTI_KEYVALUE, which requires only one SPLIT parsing on the content. This improves job performance by 50% to 100%.

- Use the LIKE operator with caution
  - To match records that start with the specified content, use `LIKE 'xxx%'`.
  - To match records that end with the specified content, use `LIKE '%xxx'`.
  - To match records that contain the specified content, use `LIKE '%xxx%'`.
  - To match records that are the same as the specified content, use `LIKE 'xxx'`, which is equivalent to `str = 'xxx'`.
  - To match an underscore (`_`), use `LIKE '%seller/id%' ESCAPE '/'`. The underscore (`_`) is escaped because it is a single-character wildcard in SQL and can match any characters. If you use `LIKE '%seller_id%'`, a lot of results are returned, such as `seller_id`, `seller#id`, `sellerxid`, and `seller1id`. These results may be unsatisfactory.
- Avoid the use of regular expressions
  Running regular expressions can be time-consuming and may require a hundred more times of computing resources in comparison with other operations such as plus, minus, multiplication, and division. If you run regular expressions under some particular circumstances, your job may be stuck in an infinite loop. Therefore, use the LIKE operator whenever possible. For information about common regular expressions, click the following related link:
  - REGEXP
  - REGEXP_EXTRACT
  - REGEXP_REPLACE

Optimize network transmission

Common partitioner policies include:

- KeyGroup/Hash: distributes data based on specified keys.
- Rebalance: distributes data to each channel by using round-robin scheduling.
- Dynamic-Rebalance: dynamically distributes data to channels with lower load based on the load status of output channels.
- Forward: similar to Rebalance if keys and channels are unchained. If keys and channels are chained, Realtime Compute for Apache Flink distributes data under specified keys to the related channels.
- Rescale: distributes data in one-to-many or many-to-one mode between input and output channels.
- Use Dynamic-Rebalance to replace Rebalance

  When you use Dynamic-Rebalance, Realtime Compute for Apache Flink writes data to subpartitions with lower load based on the amount of buffered data in each subpartition so that it can achieve dynamic load balancing. Compared with the static Rebalance policy, Dynamic-Rebalance can balance the load and improve the overall job performance when the computing capacity of output computing nodes is unbalanced. If you find the load of output nodes is unbalanced when you use Rebalance, you may prefer to use Dynamic-Rebalance. To use Dynamic-Rebalance, set the `task.dynamic.rebalance.enabled` parameter to true. The default value is false.
- Use Rescale to replace Rebalance

  > **Note** Rescale is supported only in Realtime Compute for Apache Flink V2.2.2 or later.

Assume that you have 5 parallel input nodes and 10 parallel output nodes. If you use Rebalance, each input node distributes data to all 10 output nodes by using round-robin scheduling. If you use Rescale, each input node only needs to distribute data to two output nodes by using round-robin scheduling. This reduces the number of channels, increases the buffering speed of each subpartition, and therefore improves the network efficiency. When input data is even and the numbers of parallel input and output nodes are the same, you can use Rescale to replace Rebalance. To use Rescale, set the `enable.rescale.shuffling` parameter to true. The default value is false.

Recommended configuration

In summary, we recommend that you use the following job configuration:

```
# Exactly-once semantics.
blink.checkpoint.mode=EXACTLY_ONCE
# The checkpoint interval in milliseconds.
blink.checkpoint.interval.ms=180000
blink.checkpoint.timeout.ms=600000
# Realtime Compute for Apache Flink V2.X uses Niagara as the state backend and uses it
to set the lifecycle (in milliseconds) of the state data.
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
# Realtime Compute for Apache Flink V2.X enables micro-batch processing with an interva
l of five seconds.
blink.microBatch.allowLatencyMs=5000
# The allowed latency for a job.
blink.miniBatch.allowLatencyMs=5000
# The size of a batch.
blink.miniBatch.size=20000
# Enable local aggregation. This feature is enabled by default in Realtime Compute for
Apache Flink V2.X, but you must manually enable it if you use Realtime Compute for Apac
he Flink V1.6.4.
blink.localAgg.enabled=true
# Enable PartialFinal to resolve data hotspot issues when you execute the COUNT DISTINC
T function in Realtime Compute for Apache Flink V2.X.
blink.partialAgg.enabled=true
# Enable UNION ALL for optimization.
blink.forbid.unionall.as.breakpoint.in.subsection.optimization=true
# Enable OBJECT REUSE for optimization.
#blink.object.reuse=true
# Configure garbage collection for optimization. (You cannot set this parameter if you
use a Log Service source table.)
blink.job.option=-yD heartbeat.timeout=180000 -yD env.java.opts='-verbose:gc -XX:NewRat
io=3 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4'
# Specify the time zone.
blink.job.timeZone=Asia/Shanghai
```

6.6.3. Performance optimization by using automatic configuration

To improve user experience, Realtime Compute allows you to use automatic configuration to optimize job performance.

> **Note** Automatic configuration applies to Blink 1.0 and Blink 2.0.

Background and scope

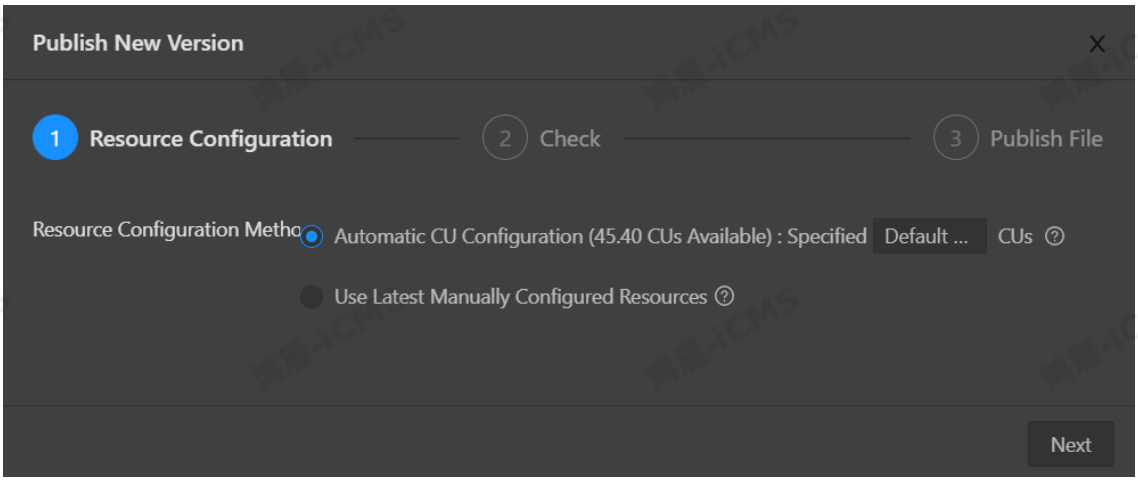
If all the operators and both the upstream and downstream storage systems of your Realtime Compute job meet the performance requirements and remain stable, automatic configuration can help you properly adjust job configurations, such as operator resources and parallelism. It also helps optimize your job throughout the entire process to resolve performance issues such as low throughput or upstream and downstream backpressure.

In the following scenarios, you can use this feature to optimize job performance but cannot eliminate job performance bottlenecks. To eliminate the performance bottlenecks, manually configure the resources or contact the Realtime Compute support team.

- Performance issues exist in the upstream or downstream storage systems of a Realtime Compute job.
  - Performance issues in the data source, such as insufficient DataHub partitions or Message Queue (MQ) throughput. In this case, you must increase the partitions of the relevant source table.
  - Performance issues in a data sink, such as a deadlock in ApsaraDB RDS.
- Performance issues of user-defined extensions (UDXs) such as user-defined functions (UDFs), user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs) in your Realtime Compute job.

Description

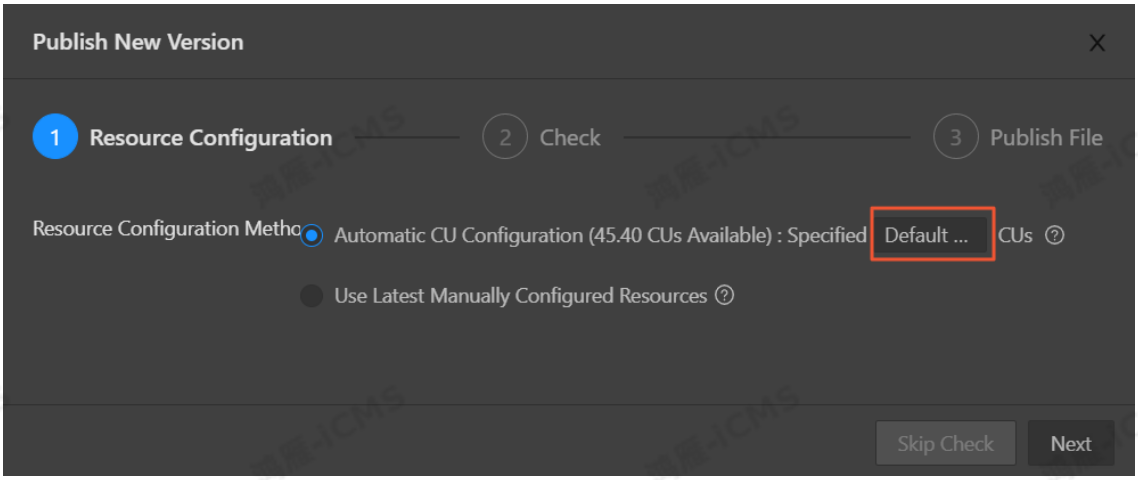
- New jobs
  i. Publish a job.
     a. After you complete SQL development and syntax check on the **Development** page, click **Publish**. The **Publish New Version** dialog box appears.



b. Specify **Resource Configuration Method**.
   - **Automatic CU Configuration**: If you select this option, you can specify the number of compute units (CUs). The automatic configuration algorithm generates an optimized resource configuration and assigns a value for the number of CUs based on the default configuration. If you use automatic CU configuration for the first time, the default number of CUs is used. This algorithm generates an initial configuration based on empirical data when you use automatic CU configuration for the first time. We recommend that you select Automatic CU Configuration if your job has been running for 5 to 10 minutes and its metrics, such as source RPS, remain stable for 2 to 3 minutes. You can obtain the optimal configuration after you repeat the optimization process for three to five times.
   - **Use Latest Manually Configured Resources**: The latest saved resource configuration is used. If the latest resource configuration is generated based on automatic CU configuration, the latest resource configuration is used. If the latest resource configuration is obtained based on the manual configuration, the manual configuration is used.

ii. Use the default configuration to start the job.

a. Use the default configuration to start the job, as shown in the following figure.



b. On the Administration page, find the job and click **Start** in the Actions column to start the job.

Assume that the default number of CUs generated the first time is 71.

> **Note** Make sure that your job runs longer than 10 minutes and its metrics such as source RPS remain stable for 2 to 3 minutes before you select Automatic CU Configuration for Resource Configuration Method.

iii. Use the automatic CU configuration to start a job.

a. Resource performance optimization

If you select Automatic CU Configuration for Resource Configuration Method and specify 40 CUs to start your job, you can change the number of CUs based on your job to optimize resource performance.

- Determine the minimum number of CUs.

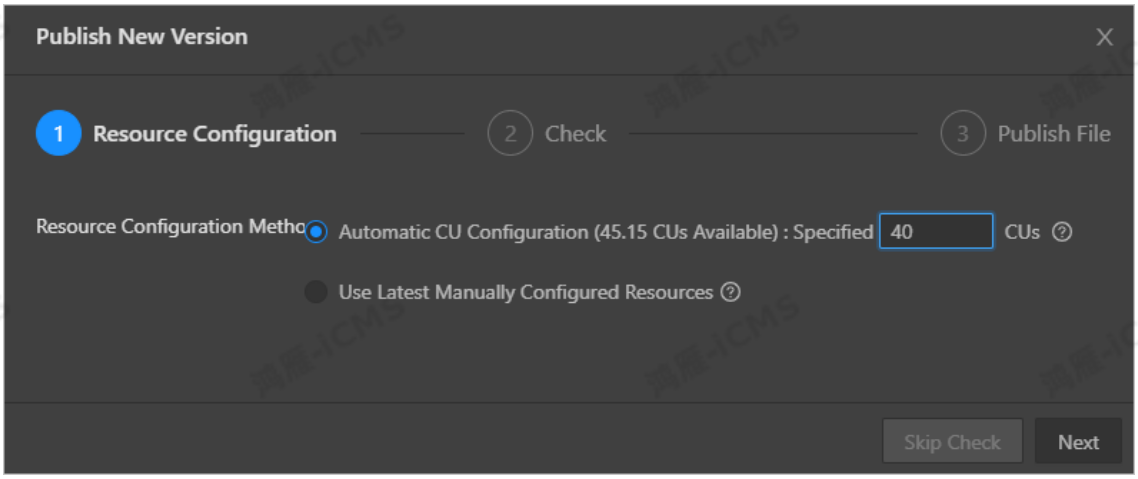
  We recommend that you set the number of CUs to a value that is greater than or equal to 50% of the default value. The number of CUs cannot be less than 1. Assume that the default number of CUs for automatic CU configuration is 71. The recommended minimum number of CUs is 36, which is calculated by using the following formula: 71 CUs × 50% = 35.5 CUs.

- Increase the number of CUs.

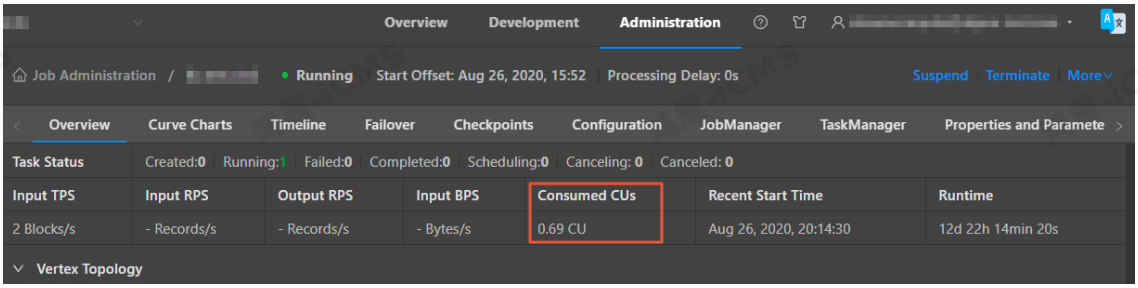
  If the throughput of your Realtime Compute job does not meet your requirements, increase the number of CUs. We recommend that you increase the number of CUs by more than 30% of the current value. For example, if the number of CUs that you specified last time is 10 CUs, you can increase the number to 13.

- Repeat the optimization process.

  If the first optimization attempt does not meet your requirements, repeat the process until you obtain the desired results. You can change the number of CUs based on your job status after each optimization attempt.



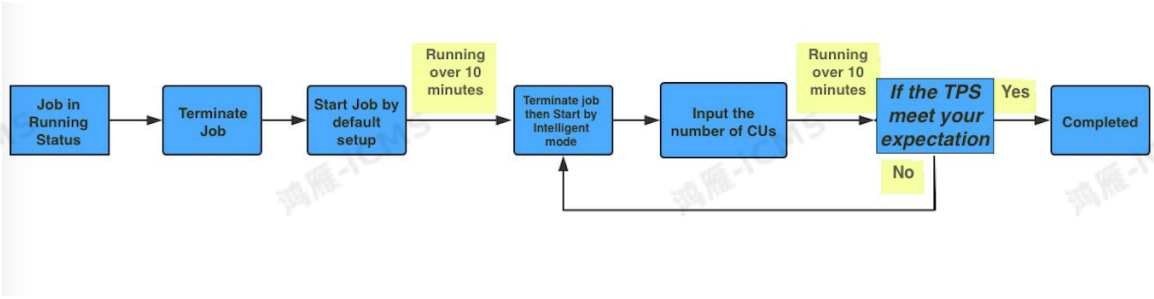
b. View the result of optimization. The following figure shows an example.



> **Note** Do not select **Use Latest Manually Configured Resources** for a new job. Otherwise, an error is returned.

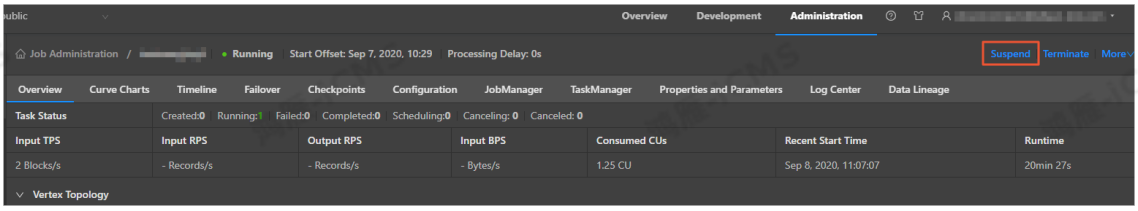
- Existing jobs

- The following figure shows the optimization process of automatic configuration.

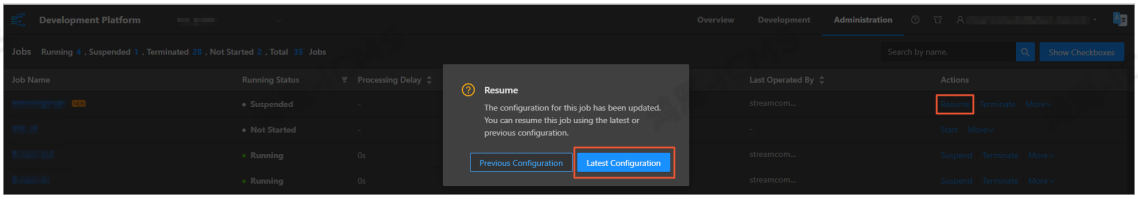


> **Note**
> - Before you use automatic configuration for a job that is in the running state, check whether stateful operations are involved. This is because the saved state data of a job may be cleared during the optimization process of automatic configuration.
> - If you make changes to a job, for example, modifying SQL statements or changing the Realtime Compute version, automatic configuration may fail. These changes may lead to topology changes, which results in some issues. For example, curve charts may not be able to display the latest data, or the state data may not able to be used for fault tolerance. In this case, resource configurations cannot be optimized based on the job running history and therefore an error is returned when you perform automatic configuration. To rectify the fault, you must treat the changed job as a new job and repeat the previous operations.

- Procedure
  a. Suspend the job.



  b. Repeat the steps performed for new jobs and resume the job with the latest configuration.



FAQ

The optimization result of automatic configuration may not be accurate in the following scenarios:

- If the job runs only for a short period of time, the data collected during data sampling is insufficient. We recommend that you increase the running duration of the job and make sure that the curves of job metrics such as source RPS remain stable for at least 2 to 3 minutes.
- A job fails. We recommend that you check and fix the failure.
- Only a small amount of data is available for a job. We recommend that you retrieve more

historical data.

- The effect of automatic configuration is affected by multiple factors. Therefore, the latest configuration obtained by using automatic configuration may not be optimal. If the effect of automatic configuration does not meet your requirements, you can manually configure the resources. For more information, see Optimize performance by manual configuration.

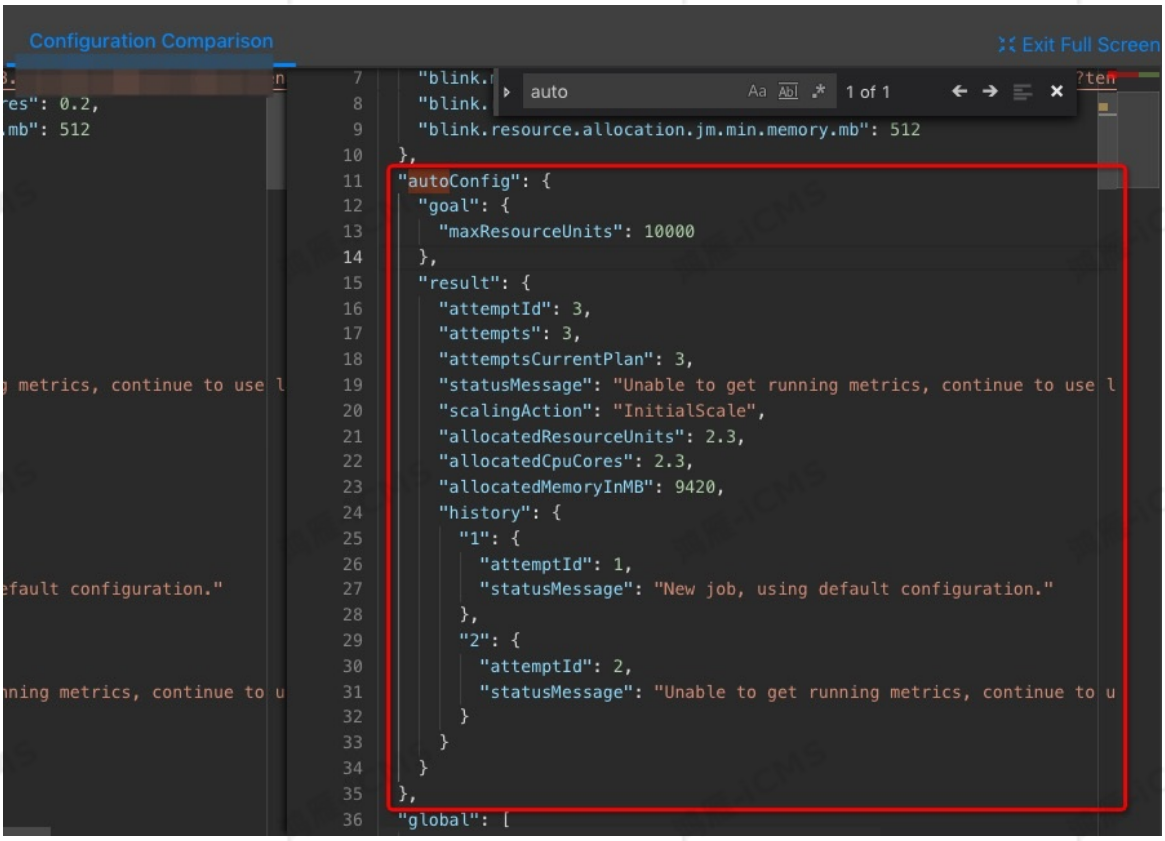
Recommendations

- To help automatic configuration accurately collect the runtime metric information of a job, make sure that the job runs stably for more than 10 minutes before you apply automatic configuration to the job.
- Job performance can be improved after you use automatic configuration for three to five times.
- When you use automatic configuration, you can specify the start offset to retrieve historical data or even accumulate large amounts of data for a job to create backpressure to accelerate the optimization effect.

Method used to determine the effectiveness of automatic configuration

Automatic configuration of Realtime Compute is enabled based on a JSON configuration file. After you use automatic configuration to optimize a job, you can view the JSON configuration file to check whether the feature is running as expected.

- You can view the JSON configuration file by using one of the following methods:
  i. View the file on the job edit page, as shown in the following figure.



  ii. View the file on the Job Administration page, as shown in the following figure.

- JSON configuration description

```
"autoconfig" : {
    "goal": {  // The goal of automatic configuration.
        "maxResourceUnits": 10000.0,  // The maximum number of CUs for a Blink job. This value cannot be changed. Therefore, you can ignore this item when you check whether the feature is running as expected.
        "targetResoureUnits": 20.0  // The number of CUs that you specified. The specified number of CUs is 20.
    },
    "result" : {  // The result of automatic configuration. We recommend that you pay attention to this item.
      "scalingAction" : "ScaleToTargetResource",  // The action of automatic configuration. *
      "allocatedResourceUnits" : 18.5, // The total resources allocated by automatic configuration.
      "allocatedCpuCores" : 18.5,      // The total CPU cores allocated by automatic configuration.
      "allocatedMemoryInMB" : 40960    // The total memory size allocated by automatic configuration.
      "messages" : "xxxx"  // We recommend that you pay attention to these messages. *
    }
}
```

  - scalingAction: If the value of this parameter is `InitialScale`, this is the first time that you use automatic configuration. If the value of this parameter is `ScaleToTargetResource`, this is not the first time that you use automatic configuration.

- If no message appears, automatic configuration runs properly. If some messages appear, you must analyze these messages. Messages are categorized into the following two types:
  - Warning: This type of message indicates that automatic configuration runs properly but you must pay attention to potential issues, such as insufficient partitions in a source table.
  - Error or exception: This type of message indicates that automatic configuration failed. The following error message is usually displayed: `Previous job statistics and configuration will be used`. The automatic configuration for a job fails in the following two scenarios:
    - The job or Blink version is modified before you use automatic configuration. In this case, the previous running information cannot be used for automatic configuration.
    - An error message that contains "exception" is reported when you use automatic configuration. In this case, you must analyze the error based on the job running information and logs.

Error messages

IllegalStateException

If the following error messages are displayed, the state data cannot be used for fault tolerance. To resolve this issue, terminate the job, clear its state, and then specify the start offset to re-read the data.

If you cannot migrate the job to a backup node, perform the following steps to mitigate the negative impact of service interruption: Roll back the job to an earlier version and specify the start offset to re-read the data during off-peak hours. To roll back the job, click **Versions** on the right side of the **Development** page. On the page that appears, move the pointer over More in the Actions column, click Compare, and then click Roll Back to Version.

```
java.lang.IllegalStateException: Could not initialize keyed state backend.
    at
org.apache.flink.streaming.api.operators.AbstractStreamOperator.initKeyedState(AbstractSt
amOperator.java:687)
    at
org.apache.flink.streaming.api.operators.AbstractStreamOperator.initializeState(AbstractS
eamOperator.java:275)
    at
org.apache.flink.streaming.runtime.tasks.StreamTask.initializeOperators(StreamTask.java:8
)
    at
org.apache.flink.streaming.runtime.tasks.StreamTask.initializeState(StreamTask.java:856)

    at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:292)
    at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
    at java.lang.Thread.run(Thread.java:834)
Caused by: org.apache.flink.api.common.typeutils.SerializationException: Cannot seriali
ze/deserialize the object.
    at
com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSecondaryState.deserializeSta
Entry(AbstractRocksDBRawSecondaryState.java:167)
    at
com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRestoreOperation.restoreRawSt
eData(RocksDBIncrementalRestoreOperation.java:425)
    at
com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRestoreOperation.restore(Rock
BIncrementalRestoreOperation.java:119)
```

```
    at
com.alibaba.blink.contrib.streaming.state.RocksDBKeyedStateBackend.restore(RocksDBKeyedSt
eBackend.java:216)
    at
org.apache.flink.streaming.api.operators.AbstractStreamOperator.createKeyedStateBackend(A
tractStreamOperator.java:986)
    at
org.apache.flink.streaming.api.operators.AbstractStreamOperator.initKeyedState(AbstractSt
amOperator.java:675)
    ... 6 more
Caused by: java.io.EOFException
    at java.io.DataInputStream.readUnsignedByte(DataInputStream.java:290)
    at org.apache.flink.types.StringValue.readString(StringValue.java:770)
    at
org.apache.flink.api.common.typeutils.base.StringSerializer.deserialize(StringSerializer.
va:69)
    at
org.apache.flink.api.common.typeutils.base.StringSerializer.deserialize(StringSerializer.
va:28)
    at
org.apache.flink.api.java.typeutils.runtime.RowSerializer.deserialize(RowSerializer.java:
9)
    at
org.apache.flink.api.java.typeutils.runtime.RowSerializer.deserialize(RowSerializer.java:
)
    at
com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSecondaryState.deserializeSta
Entry(AbstractRocksDBRawSecondaryState.java:162)
    ... 11 more
```

6.6.4. Performance optimization by using auto scaling

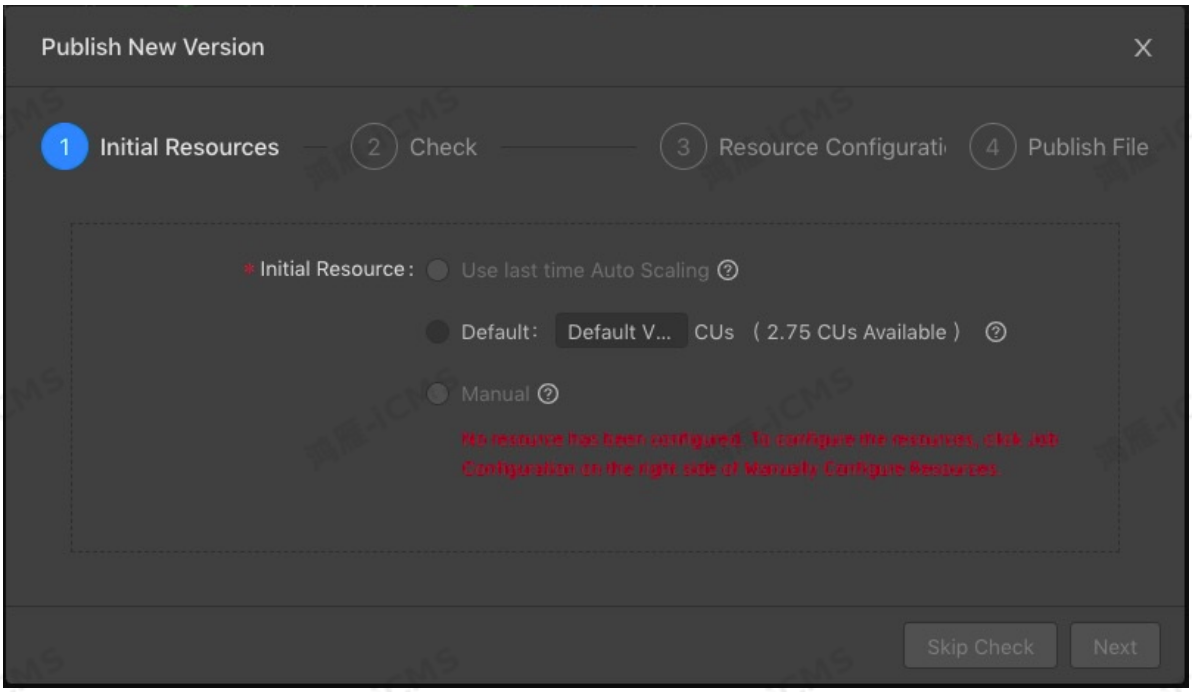
Realtime Compute for Apache Flink earlier than V3.0.0 provides AutoConf to improve job performance. However, AutoConf requires you to frequently restart the job. Realtime Compute for Apache Flink V3.0.0 and later versions support auto scaling to resolve this issue. After you start a job, Realtime Compute for Apache Flink adjusts the job configuration to reach the preset performance goal based on resource configuration rules. This process does not require any manual operations.

> **Note**
> - Auto scaling is supported only in Realtime Compute for Apache Flink V3.0.0 and later.
> - Before you upgrade Realtime Compute for Apache Flink to V3.0.0, delete all PlanJSON files generated in Realtime Compute for Apache Flink earlier than V3.0.0 and reacquire configuration files.

Enable auto scaling

You can enable auto scaling when you publish a job.

1. Go to the job editing page on the Realtime Compute for Apache Flink development platform.
   i. Log on to the Realtime Compute development platform.
   ii. In the top navigation bar, click **Development**.
   iii. On the **Development** page that appears, double-click the target job, which may be nested under a folder, to go to the job development page.
2. In the upper part of the job editing section, click **Publish** to go to the **Publish New Version** page.
3. In the **Initial Resources** step, select a resource type and click **Next**.



   - **Use last time Auto Scaling**: uses the PlanJSON file for the latest auto scaling to start the job. You can select **Use last time Auto Scaling** when the following conditions are met:
     - The job is published with auto scaling enabled and uses the latest configuration.
     - The job is in the **Suspended** state.

- The auto scaling configuration is obtained. To obtain the auto scaling configuration, click **Configurations** on the right, move the pointer to Configurations in the upper-right corner, and select **Acquire Auto Scaling Configuration**.

  - **Default**: uses the default resource configuration to start the job. You can select this option to publish a new job or an existing job whose logic is not modified and compatible with Realtime Compute for Apache Flink.
  - **Manual**: uses manually configured resource configuration to start the job. Select this option to manually configure resources or modify the **auto scaling configuration**.

4. In the **Check** step, check the job and click **Next**.
5. In the **Resource Configuration** step, configure **auto scaling** parameters and click **Next**.

Parameter	Description
Automatic Scaling	Specifies whether to enable auto scaling. Select **ON**.
Maximum number of PlanJson CUs	The maximum available CUs for the job. One CU consists of **1 CPU core and 4 GB of memory**. The value of this parameter must be less than the number of available CUs in the project.

Optimization Policy	The policy for optimizing the job configuration. Valid value:**Data Pending Time**. Realtime Compute for Apache Flink optimizes the job configuration based on **Optimization Policy** and **Expected Value**.
Expected Value	The threshold of the data pending time, in seconds. If data from the data source is pending for a period of time that exceeds the threshold, Realtime Compute for Apache Flink triggers auto scaling to adjust the parallelism for the job.

> **Note** For example, set Expected Value to 5. If data from the data source is pending for more than 5 seconds, Realtime Compute for Apache Flink keeps reducing the parallelism for the job until the pending time is shortened to less than 5 seconds.

6. In the **Publish File** step, click **Publish**.
7. Start the job. For more information about this issue, see Start a job.

Disable auto scaling

> **Note** You can disable auto scaling for a job only if auto scaling is enabled for the job when the job is published.

You can disable auto scaling for a job in the Running state.

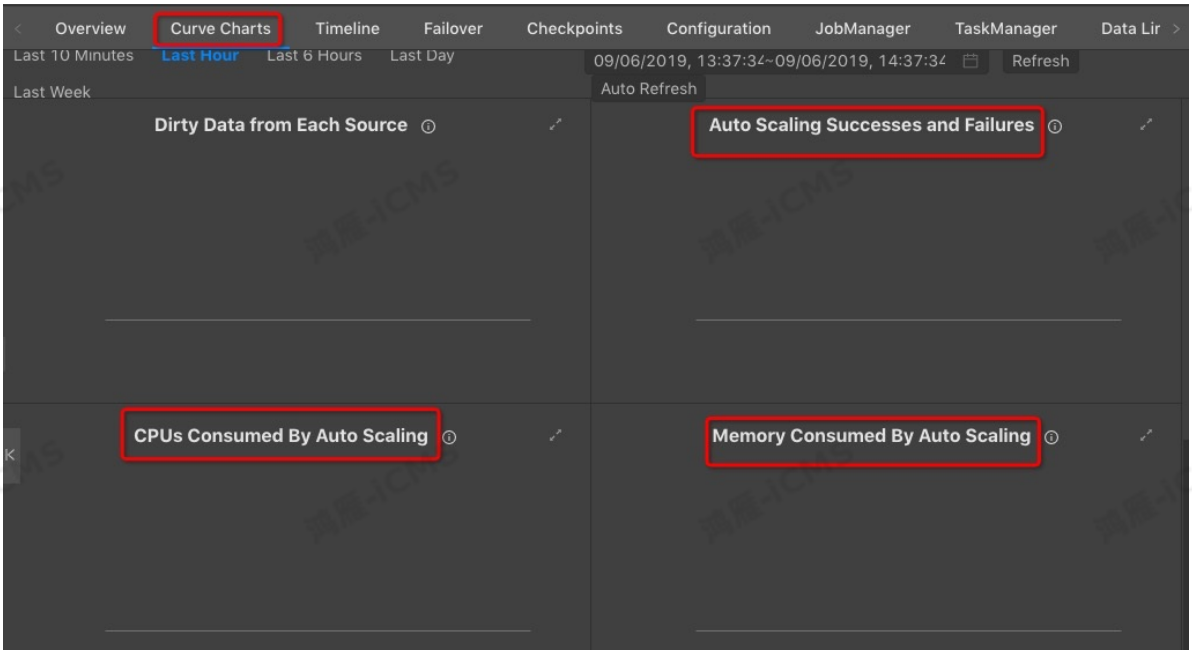
1. Go to the **Job Administration** page on the Realtime Compute for Apache Flink development platform.
   i. Log on to the Realtime Compute development platform.
   ii. In the top navigation bar, click **Administration**.
   iii. On the **Jobs** page that appears, click the target job name under the **Job Name** field.
2. Click **Disable Auto Scaling** in the upper-right corner.
3. Click **Confirm**.

> **Note** The actions in the **Auto Configuration** column on the **Job Administration** page are available only if auto scaling is enabled when you publish the job.

View information about auto scaling

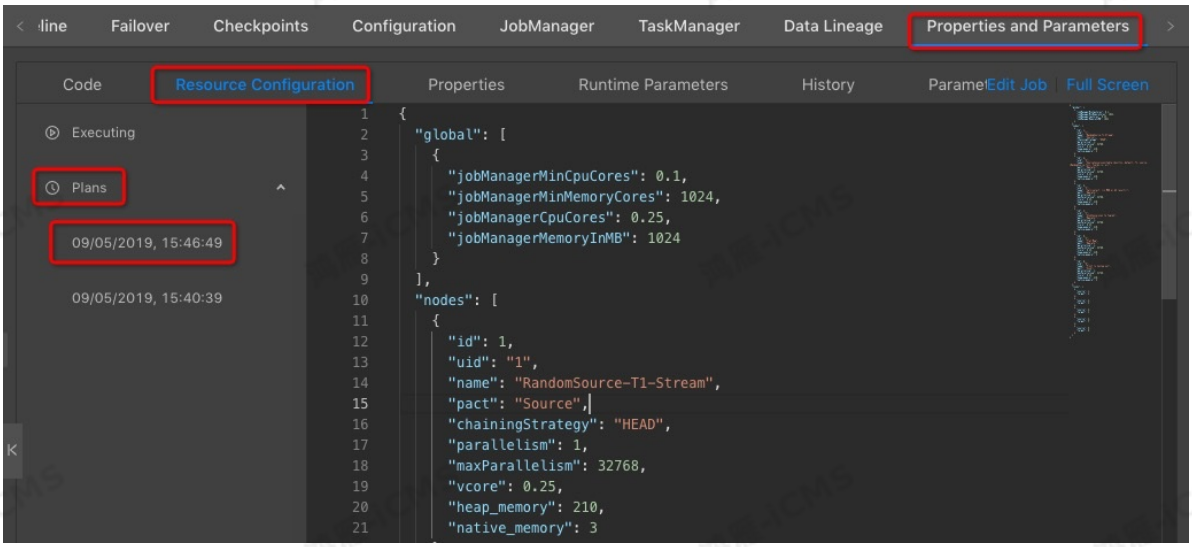
> **Note** Go to the **Job Administration** page. To go to the **Job Administration** page, perform the following steps:
>
> 1. Log on to the Realtime Compute development platform.
> 2. In the top navigation bar, click **Administration**.
> 3. On the **Jobs** page that appears, click the target job name under the **Job Name** field.

- AutoScale Metric
  Navigate to **Curve Charts > Overview** to view information about auto scaling.



Metric	Description
Auto Scaling Successes and Failures	The number of successful and failed auto scaling operations
CPUs Consumed By Auto Scaling	The CPU resources used for auto scaling.
Memory Consumed By Auto Scaling	The memory resources used for auto scaling.

- Information about the PlanJSON file generated for auto scaling
  On the Job Administration page, navigate to **Properties and Parameters > Resource Configuration > Plans**. Select the required version to view information about the PlanJSON file that is generated for auto scaling.



FAQ

- Q: What do I do if auto scaling cannot be triggered?

  A: To troubleshoot this issue, perform the following steps:

- Check whether the job frequently fails. To trigger auto scaling, make sure that the job is in correct logic and runs in a stable manner.
- View **JobManager** logs to check whether system exceptions occur.

- Q: What do I do if auto scaling does not take effect?
  A: To troubleshoot this issue, perform the following steps:

i. Check whether the resources consumed by the job reach the upper limit.

ii. Check the source node logic to determine whether excessive operators are connected to the source node. If excessive operators are connected to the source node, edit the PlanJSON file to remove some operators. For more information, see Optimize performance by manual configuration.

iii. View **JobManager** logs to check whether system exceptions occur.

- Q: What issues might I run into if I enable auto scaling?
  A:
  - The job is automatically restarted.

    If you enable auto scaling, Realtime Compute for Apache Flink adjusts the parallelism and resources based on the number of data streams. The job may automatically restart the job to adjust resources when the data streams increase or decrease.
  - Data transmission is delayed within a short period of time.

    When the data streams enter the off-peak period, Realtime Compute for Apache Flink triggers auto scaling to reduce the parallelism and resources. When the data streams increase, the resources may become insufficient for data transmission, which causes delay within a short period of time.
  - The job cannot be resumed.

    In some scenarios, auto scaling does not work, and the job may be delayed. Realtime Compute for Apache Flink has to frequently adjust the job configuration. As a result, the job cannot be resumed.
  - The parallelism is reduced and then increased.

    For a job that uses window functions or aggregate functions, when the state data increases, the performance for accessing the state data degrades, and the parallelism is reduced when the job is started. When the job is running, the parallelism increases as the state data accumulates until the amount of the state data becomes steady.

6.6.5. Optimize performance by manual configuration

You can optimize the performance of a Realtime Compute for Apache Flink job by adjusting the settings of job, resource, and upstream and downstream storage parameters.

Overview

You can configure the following types of parameters to optimize job performance:

- Upstream and downstream storage parameters
- Job parameters, such as **miniBatch**
- Resource parameters, such as parallelism, core, and heap_memory

This topic describes how to configure the preceding three types of parameters. After you reconfigure parameters for a job, you must terminate and then start the job, or suspend and then resume the job to apply new settings. For more information, see Apply new configurations.

Upstream and downstream storage parameters

In Realtime Compute for Apache Flink, each data record can trigger read and write operations on the source and result tables. This affects the performance of upstream and downstream storage resources. To address this performance issue, you can configure batch size parameters to specify the number of data records that can be read from a source table or written to a result table at a time. The following table describes the source and result tables that support batch size parameters.

Table	Parameter	Description	Value
DataHub source table	batchReadSize	The maximum number of data records that can be read at a time.	Optional. Default value: 10.
DataHub result table	batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 300.
Log Service source table	batchGetSize	The maximum number of log items that can be read from a log group at a time.	Optional. Default value: 100.
AnalyticDB for MySQL V2.0 result table	batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 1000.
ApsaraDB RDS result table	batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 4096.

> **Note** To configure the batch data read and write feature, you can add the preceding parameters to the WITH clause in a DDL statement for a storage system. For example, add `batchReadSize='<number>'` to the WITH clause.

Job parameters

The **miniBatch** parameter can be used only to optimize the GROUP BY operator. If you use Flink SQL to process streaming data, Realtime Compute for Apache Flink reads state data each time a data record arrives. This consumes a large number of I/O resources. If you configure the **miniBatch** parameter, Realtime Compute for Apache Flink uses the same key to read the state data only once for the data records and generates only the latest data record. This reduces the frequency to read the state data and minimizes downstream data updates. You can configure the **miniBatch** parameter based on the following rules:

- After you add parameters for a job, terminate and then start the job to apply the new settings.
- After you change parameter settings for a job, suspend and then resume the job to apply the new settings.

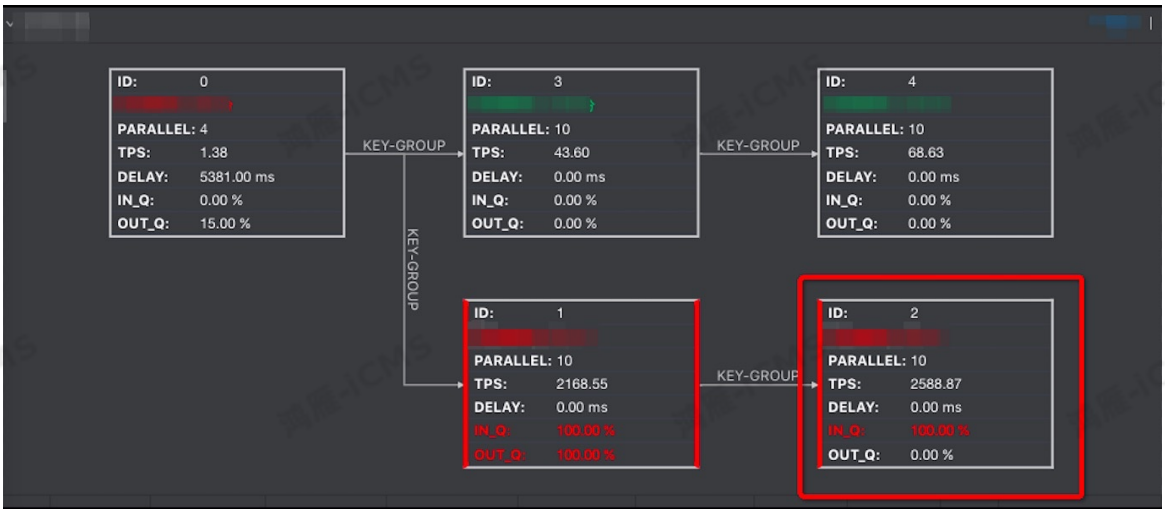
```
# Enable window miniBatch in Realtime Compute for Apache Flink V3.2 and later. By defau
lt, window miniBatch is disabled in Realtime Compute for Apache Flink V3.2 and later.
sql.exec.mini-batch.window.enabled=true
# Use the exactly-once semantics.
blink.checkpoint.mode=EXACTLY_ONCE
# Specify the checkpoint interval. Unit: milliseconds.
blink.checkpoint.interval.ms=180000
blink.checkpoint.timeout.ms=600000
# Use Niagara as the state backend to configure time-to-live (TTL) for the state backen
d in Realtime Compute for Apache Flink V2.0 and later. The unit of TTL is milliseconds.

state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
# In Realtime Compute for Apache Flink V2.0 and later, enable micro-batch processing th
at is performed at an interval of 5 seconds. You cannot configure this parameter when y
ou use a window function.
blink.microBatch.allowLatencyMs=5000
# Specify the latency that is allowed for a job.
blink.miniBatch.allowLatencyMs=5000
# Enable miniBatch for the node that joins two streams.
blink.miniBatch.join.enabled=true
# Specify the size of a batch of data.
blink.miniBatch.size=20000
# Enable local aggregation. By default, this feature is enabled in Realtime Compute for
Apache Flink V2.0 and later. If you use Realtime Compute for Apache Flink V1.6.4, you m
ust manually enable this feature.
blink.localAgg.enabled=true
# Enable partial aggregation to improve efficiency when you run the CountDistinct funct
ion in Realtime Compute for Apache Flink V2.0 and later.
blink.partialAgg.enabled=true
# Enable UNION ALL for optimization.
blink.forbid.unionall.as.breakpoint.in.subsection.optimization=true
# Configure garbage collection for optimization. You cannot configure this parameter wh
en you use a Log Service source table.
blink.job.option=-yD heartbeat.timeout=180000 -yD env.java.opts='-verbose:gc -XX:NewRat
io=3 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4'
# Specify the time zone.
blink.job.timeZone=Asia/Shanghai
```

Resource parameters

To optimize resource configurations, perform the following steps:

1. Issue analysis

i. In the following topology, the percentage of the input queues at Task Node 2 reaches 100%. The data at Task Node 2 is stacked up and causes backpressure on Task Node 1. At Task Node 1, the percentage of the output queues has reached 100%.



ii. Click Task Node 2.

iii. In the Vertex Topology section of the Overview tab, click the **SubTask List** tab. Then, find the subtask in which the value of **In Queue** is `100%`.

iv. Click **LOG 0** in the ID column in the row of the subtask.

v. On the **Metrics Graph** tab, view the CPU and memory usage.

2. Performance

i. On the right side of the job editing page, click the **Configurations** tab to view the details about resource configurations.

ii. On the page that appears, change the parameter values of one or more operators in a group.

- To change the parameter values of one operator, perform the following steps:
  a. In the **GROUP** box, click the plus sign (+) in the upper-right corner.
  b. Move the pointer over the Operator box.
  c. Click the  icon next to the operator name.
  d. In the **Modify Operator Data** dialog box, change the parameter values and click OK.
- To change the parameter values of multiple operators in a group at a time, perform the following steps:
  a. Move the pointer over the **GROUP** box.
  b. Click the  icon next to **GROUP**.
  c. In the **Modify Operator Data** dialog box, change the parameter values based on your business requirements and click OK.

iii. In the upper-right corner of the **Configurations** page, choose **Configurations > Apply**.

- If the job performance is not significantly improved after you change the values of the resource parameters for the group, perform the following steps to troubleshoot the issue:
  a. Check whether data skew exists on the operator.
  b. Check whether subtasks of complex operators, such as GROUP BY, WINDOW, and JOIN, are running as expected.
- To remove an operator from a chain, perform the following steps:
  a. Click the operator that you want to remove.
  b. In the Modify Operator Data dialog box, set **chainingStrategy** to `HEAD`.
  If the **chainingStrategy** parameter of this operator is set to `HEAD`, you must also set the **chainingStrategy** parameter to `HEAD` for the next operator. The following table describes the valid values of the **chainingStrategy** parameter.

Parameter	Description
ALWAYS	Operators are combined to increase the parallelism and optimize job performance.
NEVER	Operators are not combined with the upstream and downstream operators.
HEAD	Operators are combined with only the downstream operators.

3. Rules and suggestions
   - We recommend that you set `core:heap_memory` to 1:4. This indicates that each CPU core is assigned 4 GB of memory. Examples:
     - If the **core** parameter of operators is set to 1 and the **heap_memory** parameter of the operator is set to 3, the system assigns **1 compute unit (CU) and 4 GB** of memory to the chain.
     - If the **core** parameter of operators is set to 1 and the **heap_memory** parameter of operators is set to 5, the system assigns **1.25 CUs and 5 GB** of memory to the chain.

     > **Note**
     > - The total number of cores for an operator is calculated by using the following formula: **Value of the parallelism parameter × Value of the core parameter**.
     > - The total heap_memory size for an operator is calculated by using the following formula: **Value of the parallelism parameter × Value of the heap_memory parameter**.
     > - The core value for a chain is the maximum core value among the operators in the group. The heap_memory size for a chain is the total heap_memory size of all the operators in the chain.

   - **parallelism**

- Source node
  The number of source nodes is a multiple of the number of upstream partitions. For example, if the number of source nodes is 16, you must set the parallelism parameter to a divisor of 16, such as 16, 8, or 4. The divisor must exclude 16.

  > **Note** The value of the parallelism parameter for the source nodes cannot exceed the number of shards for the source nodes.

- Operator node
  Specify the parallelism parameter of the operator nodes based on the estimated queries per second (QPS).
  - If the QPS is low, you can set the number of operator nodes to the value that is the same as the parallelism of the source nodes.
  - If the QPS is high, make sure that the number of operator nodes is greater than the parallelism of the source nodes. For example, if the parallelism is 16, set the number of operator nodes to a value that is greater than 16, such as 64, 128, or 256.
- Sink node
  Set the parallelism parameter of the sink nodes to a value that is two to three times the number of downstream partitions.

  > **Note** Do not set the parallelism parameter of the sink nodes to a value that is greater than three times the number of downstream partitions. Otherwise, write timeout or failures may occur. For example, if the number of sink nodes is 16, do not set the parallelism parameter of these sink nodes to a value that is greater than 48.

- **core**

  This parameter specifies the number of CPU cores. You can specify this parameter based on the actual CPU utilization. The recommended value of this parameter is 0.25. The default value is 0.1.
- **heap_memory**

  The heap memory size. Unit: MB. You can configure this parameter based on the actual memory usage. The default value is 256.
- **state_size**
  You must set the **state_size** parameter to `1` for task nodes where the GROUP BY, JOIN, OVER, or WINDOW operators are used. This way, the system assigns extra memory for the operator to access state data. The default value of the **state_size** parameter is 0.

  > **Note** If you do not set **state_size** to `1`, the job may fail.

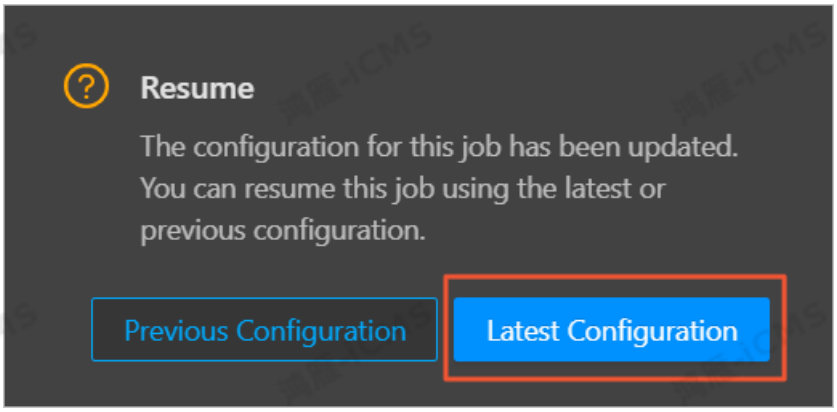
Apply new configurations

After you configure the parameters, we recommended that you suspend and then resume the job, but not terminate and then start the job. This ensures that the configurations take effect. The job status is cleared when the job is terminated. This may change execution results.

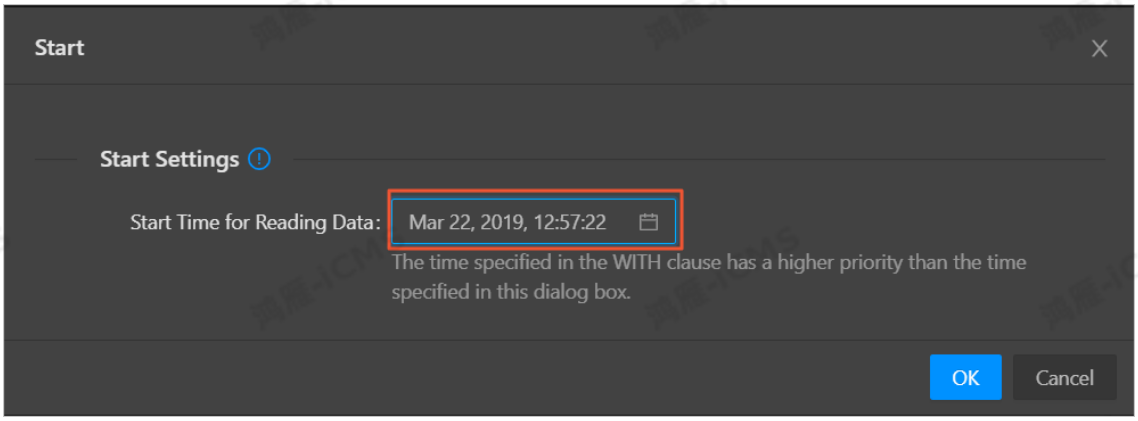
> **Note**
> - You can suspend and then resume a job after you change the values of the resource parameters, parameters in the WITH clause, or job parameters.
> - You can terminate and then start a job after you modify the SQL logic, change the job version, add parameters to the WITH clause, or add job parameters.

After you restart or resume the job, you can click the **Overview** tab on the **Administration** page and click the **Vertex Topology** tab to check whether the new configurations take effect.

- To suspend and resume a job, perform the following steps:
  i. Publish a job. For more information, see Publish a job. Set **Resource Configuration Method** to **Use Latest Manually Configured Resources**.
  ii. On the **Administration** page, find the job that you want to suspend and click **Suspend** in the Actions column.
  iii. On the **Administration** page, find the job that you want to resume and click **Resume** in the Actions column.
  iv. In the **Resume** dialog box, click **Latest Configuration**.



- To terminate and then start a job, perform the following steps:
  i. Terminate a job.
    a. Log on to the Realtime Compute development platform.
    b. In the top navigation bar, click **Administration**.
    c. On the **Administration** page, find the job that you want to terminate, and click **Terminate** in the **Actions** column.
  ii. Start the job.
    a. Log on to the Realtime Compute development platform.
    b. In the top navigation bar, click **Administration**.
    c. On the **Administration** page, find the job that you want to start, and click **Start** in the **Actions** column.
    d. In the **Start** dialog box, specify **Start Time for Reading Data**.



e. Click **OK**. The job is started.

> **Note** Start Time for Reading Data specifies the time when Realtime Compute for Apache Flink starts to read data from the source table.
>
> - If you select the current time, Realtime Compute for Apache Flink reads data that is generated after the current time.
> - If you select a previous time, Realtime Compute for Apache Flink reads data that is generated from the selected time. This is used to trace historical data.

Parameters

- Global

  isChainingEnabled specifies whether chaining is enabled. The default value is true. Use the default value for this parameter.

- Nodes

Parameter	Description	Allow modification
id	The unique ID of the node. The node ID is generated by the system.	No
uid	The unique user identifier (UID) of the node. The UID is used to generate the operator ID. If you do not specify this parameter, the UID is the same as the node ID.	No
pact	The node type, such as data source, operator, or data sink.	No
name	The name of the node. You can customize this parameter.	Yes
slotSharingGroup	Specifies whether subtasks can share the same slot. Use the default value for this parameter.	No
chainingStrategy	Defines the operator chaining strategy. If an operator is combined with an upstream operator, they run in the same thread. They are combined into an operator chain that has multiple running steps. Valid values: ◦ ALWAYS: Operators are combined to increase the parallelism and optimize job performance. ◦ NEVER: Operators are not combined with the related upstream or downstream operators. ◦ HEAD: Operators are combined with only the downstream operators.	Yes
parallelism	The number of parallel jobs on the node. You can increase the value based on your business requirements. Default value: 1.	Yes
core	The number of CPU cores. You can specify this parameter based on the actual CPU utilization. Default value: 0.1. Recommended value: 0.25.	Yes
heap_memory	The heap memory size. You can specify this parameter based on the memory size that needs to be used. Default value: 256 MB.	Yes

direct_memory	The non-heap memory of a Java Virtual Machine (JVM). Unit: MB. Default value: 0.	You can change the value of this parameter, but we recommend that you use the default value.
native_memory	The JVM non-heap memory that is used for the Java Native Interface (JNI). Default value: 0. You can set this parameter to 10 based on your business requirements. This memory is mainly used for state backends.	You can change the value of this parameter, but we recommend that you use the default value.

- Chain
  A Flink SQL task is a directed acyclic graph (DAG) that contains multiple nodes or operators. Some upstream and downstream operators can be combined into a new operator when the operators run in the same thread. This process is known as a chain. As a result, the total number of CPU cores for the new operator is the maximum number of CPU cores among all the operators in the chain. The memory size for the chain equals the total memory size of all the operators in the chain. An operator chain can significantly reduce data transmission costs.

  > **Note**
  > - Only operators that have the same parallelism value can be combined to form a chain.
  > - You cannot add a GROUP BY operator to a chain.

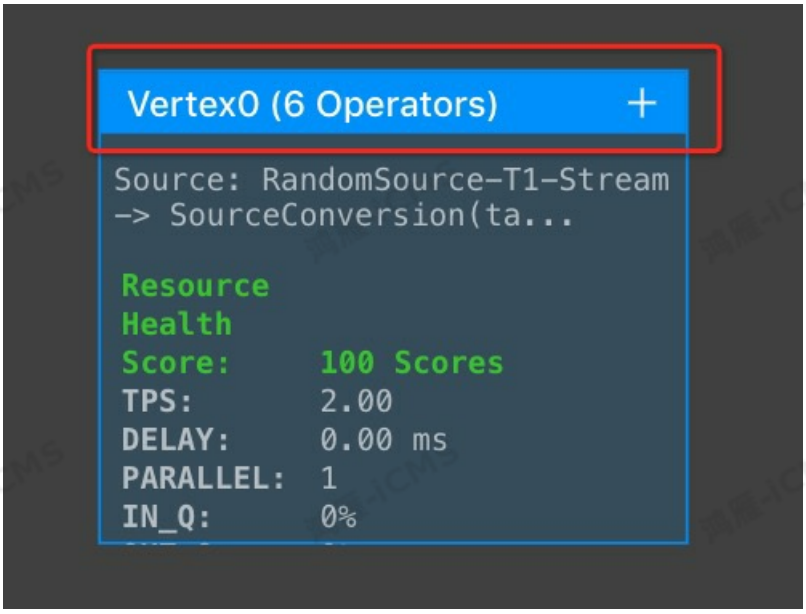
6.6.6. Typical backpressure scenarios and optimization ideas

Backpressure is an important concept in streaming shuffle. If the processing capability of downstream storage systems is insufficient, Realtime Compute notifies upstream storage systems to stop sending data to avoid data loss. In this scenario, backpressure occurs. This topic describes typical backpressure scenarios and optimization ideas.

Backpressure detection mechanism

A job backpressure detection mechanism is provided in Realtime Compute versions later than V3.0.0. With this mechanism, Realtime Compute detects congestion in the output network buffer of a vertex to determine whether backpressure exists in the vertex. A vertex is a group of operators associated as a chain. To check the backpressure for a job, follow these steps:

1. In the top navigation bar, click **Administration**.
   i. Log on to the Realtime Compute development platform.
   ii. In the top navigation bar, click **Administration**.
   iii. On the **Jobs** page that appears, click the target job name under the **Job Name** field.
2. In the left-side navigation pane, click the running job for which you want to check the backpressure. In the **Vertex Topology** section of the **Overview** tab that appears, click the blue border of the vertex that you want to check for the job.



3. In the right-side pane, click the **BackPressure** tab and view the backpressure status in the **Status** column.
   - If **high** with a red indicator is displayed, the vertex has backpressure.
   - If **ok** with a green indicator is displayed, the vertex does not have backpressure.

Backpressure scenarios and optimization ideas

> **Note** In the following vertex topology diagrams, the vertex in green indicates that no backpressure is detected, whereas the vertex in red indicates that backpressure is detected.

- **Scenario 1: Only one vertex exists and no backpressure is detected.**

Due to Flink features, no network buffer is configured on the output of the last vertex. In this case, data is directly written into downstream storage systems. If a job has only one or the last vertex, the backpressure detection fails. Therefore, this vertex topology diagram does not indicate that no backpressure is detected in the job. To further determine if and where backpressure exists, you must split the operators in Vertex 0. For more information about how to split the operators, see Resource parameters.

- **Scenario 2: Multiple vertices exist and backpressure is detected on the second to last vertex.**

This vertex topology diagram shows that Vertex 1 has backpressure and Vertex 2 has a performance bottleneck. You can check the operator names in Vertex 2 to determine the actions that you can take.

- If only write operations into downstream storage systems are involved, the backpressure may be caused by the slow writing speed. We recommend that you increase the parallelism for Vertex 2 or set the `batchsize` parameter for the result table. For more information, see Upstream and downstream storage parameters.
- If operations in addition to the write operation into downstream storage systems are involved, you must split the operators that correspond to those operations for further check. For more information about how to split the operators, see Resource parameters.

- **Scenario 3: Multiple vertices exist and backpressure is detected on a vertex other than the second to last vertex.**

This vertex topology diagram shows that Vertex 0 has backpressure and Vertex 1 has a performance bottleneck. You can check the operator names in Vertex 1 to determine the actions that you can take. The common operations and related optimization methods used in this scenario are as follows:

- GROUP BY operation: You can increase the parallelism or set the `miniBatch` parameter to optimize the state operation. For more information, see Job parameters.
- JOIN operation between dimension tables: You can increase the parallelism or set a cache policy for dimension tables. For more information, see relevant dimension table documents.
- User-defined extension (UDX) operation: You can increase the parallelism or optimize the related UDX code.

- **Scenario 4: Multiple vertices exist and no backpressure is detected on all the vertices.**

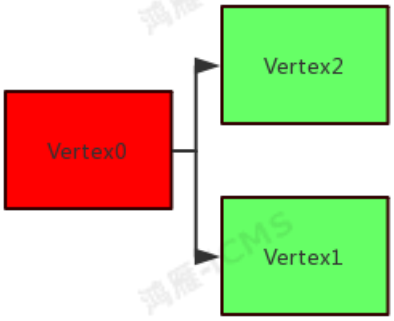
This vertex topology diagram shows that Vertex 0 has a potential performance bottleneck. You can check the operator names in Vertex 0 to determine the actions that you can take.

- If only read operations from the source table are involved, the slow reading speed causes high latency. However, Realtime Compute does not have performance bottlenecks. In this case, you can increase the parallelism of the source operator or set the `batchsize` parameter for reading the source data. For more information, see Upstream and downstream storage parameters.

  > **Note** The parallelism of the source operator cannot be greater than the number of shards of the upstream storage systems.

- If operations in addition to the read operation from the source table are involved, we recommend that you split the operators involved in other operations first. For more information about how to split operators, see Resource parameters.

- **Scenario 5: Backpressure is detected on a vertex but no backpressure is detected on its subsequent parallel vertices.**



This vertex topology diagram shows that Vertex 0 has backpressure but whether Vertex 1 or Vertex 2 has a performance bottleneck cannot be determined. You can preliminarily determine the vertex where a performance bottleneck exists based on the **IN_Q** metric of Vertex 1 and Vertex 2. The vertex whose **IN_Q** remains 100% for a long period of time may have a performance bottleneck. To further determine where the performance bottleneck exists, you must split the operators of the vertex. For more information about how to split operators, see Resource parameters.

6.6.7. SQL Tuning Advisor

6.6.7.1. Partitioned All Cache

If you use the Cache All policy to join a super-large dimension table with another table, processes may fail to load full data of the dimension table into the cache. In this case, you can use the Partitioned All Cache policy to optimize the loading performance.

Background information

When you join a dimension table with another table, you can set the cache parameter to ALL to use the Cache All policy. This policy requires that all processes load full data of the dimension table to the cache. The memory size configured for the join node must be at least twice that of the dimension table.

If the dimension table is large, the join node consumes a large amount of memory. This also increases the garbage collection overhead. If the size of a dimension table exceeds 1 TB, only partial data of the table can be loaded to the memory. Optimization by using Partitioned All Cache is introduced to resolve this issue. If Partitioned All Cache is enabled, input data is shuffled based on join keys, and each process needs to load only the required data of the dimension table to the cache.

You can set partitionedJoin to true to enable Partitioned All Cache. This reduces memory consumption. However, input data shuffling based on join keys causes additional network and CPU overheads. We recommend that you do not enable Partitioned All Cache in the following scenarios:

- Input data is severely skewed on the join keys. If you use Partitioned All Cache in this scenario, data skew slows down the running of some nodes.
- The size of the dimension table is small. For example, the size of the dimension table is less than 2 GB. If you enable Partitioned All Cache in this scenario, the memory consumption is slightly reduced, whereas high network and CPU overheads are generated.

Optimization method

Add **partitionedJoin = 'true'** to the WITH clause of the DDL statement of the dimension table.

Sample code

```
 CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
  type = 'odps',
  endPoint = 'your_end_point_name',
  project = 'your_project_name',
  tableName = 'your_table_name',
  accessId = 'your_access_id',
  accessKey = 'your_access_key',
  `partition` = 'ds=20180905',
  cache = 'ALL',
  partitionedJoin = 'true' -- Enable Partitioned All Cache.
);
```

6.6.7.2. miniBatch and microBatch

You can increase the throughput by enabling miniBatch or microBatch.

Background information

Both miniBatch and microBatch are used for micro-batch processing. If you enable miniBatch or microBatch, data processing is triggered when the data in the cache reaches a specified threshold. This reduces the frequency at which Realtime Compute for Apache Flink accesses the state data. This way, the throughput is increased and data output is reduced. However, the two methods have different triggering mechanisms:

- miniBatch triggers micro-batch processing by using the timer threads that are registered with each task. This requires some thread scheduling overheads.
- microBatch triggers micro-batch processing by using event messages, which are inserted into the data sources at a specified interval. microBatch is an enhancement of miniBatch. microBatch has higher data serialization efficiency, higher throughput, lower backpressure, and lower latency than miniBatch.

We recommend that you enable micro-batch processing in most scenarios, such as data aggregation, to improve system performance. We recommend that you do not enable micro-batch processing in the following scenarios:

- Low latency is required. Micro-batch processing achieves high throughput at the expense of latency.
- The aggregation degree of GroupAggregate is low (**O/I > 0.8**). In such scenarios, almost no data is aggregated in a batch.

Optimization method

On the **Parameters** tab of the **Development** page, set **blink.microBatch.allowLatencyMs** or **blink.miniBatch.allowLatencyMs**. The two parameters have the same effect. The unit of them is millisecond.

> **Note** If you set both parameters for a job, we recommend that you set them to the same value. If they are set to different values, the second one in the code takes effect.

Sample code

- Enable microBatch.

```
blink.microBatch.allowLatencyMs=5000
```

- Enable miniBatch.

```
blink.miniBatch.size=20000
```

- Enable microBatch and miniBatch at the same time.

```
# The maximum number of data records that can be cached in each batch. You must set t
his parameter to avoid out of memory (OOM).
blink.miniBatch.size=20000
blink.microBatch.allowLatencyMs=5000
blink.miniBatch.allowLatencyMs=6000--- This configuration takes effect.
```

6.6.7.3. Cache policy

When you join two dimension tables, you can configure a cache policy to improve the job throughput.

Background information

In Realtime Compute for Apache Flink, you can set the cache parameter to specify a cache policy:

- If the cache parameter is set to LRU, some data in the dimension table is cached. The system creates a local LRU cache for each join node. Realtime Compute for Apache Flink searches for data in the cache each time it reads a data record in the source table. The data that meets the requirement is returned. This reduces I/O requests. If no data in the cache meets the requirement, Realtime Compute for Apache Flink searches the dimension table. The data that meets the requirement is stored in the cache for subsequent queries.

  To limit the volume of data that can be stored in the cache, you can set **cacheSize**. To regularly update the data of a dimension table, you can set **cacheTTLMs** to adjust the cache expiration time. **cacheTTLMs** takes effect for all cached data records. If a cached data record is not accessed within the specified period of time, it is removed from the cache.

- If the cache parameter is set to All, all data in the dimension table is cached. The system creates an asynchronous thread for the join node to synchronize data between the cache and dimension table. The input data is blocked from the moment a job is started to the moment loading data to the cache is completed. This ensures that the data in the dimension table is loaded to the cache before input data processing starts.

  Realtime Compute for Apache Flink searches the cache in subsequent dimension table queries. If the data that meet the requirement cannot be found in the cache, the join key does not exist. If data in the cache expires, Realtime Compute for Apache Flink reloads the data in the dimension table to the cache. The reloading process does not affect the join operation of dimension tables. The reloaded data is stored in the temporary memory. The atomic substitution operation is performed after all data in the dimension table is reloaded.

  If cache is set to ALL, the join operation of dimension tables can achieve excellent performance because few I/O requests are initiated. However, the memory must be large enough to store the data of two dimension tables.

> **Note** Do not specify the cache policy if data in the cache is not allowed for each dimension table query.

Optimization method

Add **cache='LRU'** or **cache='ALL'** to the WITH clause in the DDL statement of the dimension table. The following table describes the parameters related to the cache policy.

Parameter	Description	Required	Remarks
cache	The cache policy.	No	• None: indicates that data is not cached. This is the default value. • LRU: If you set this parameter to LRU, you must configure cacheSize, cacheTTLMs, and partitionedJoin. • ALL: If you set this parameter to ALL, you must configure cacheTTLMs, cacheReloadTimeBlackList, and partitionedJoin.

cacheSize	The cache size. Unit: rows.	No	You can set this parameter only after you set the cache parameter to LRU. Default value: 10000.
cacheTTLMs	The cache expiration period or the cache reloading interval. Unit: milliseconds.	No	• If you set the cache parameter toLRU, this parameter specifies the cache expiration period. The cache does not expire by default. • If you set the cache parameter toALL, this parameter specifies the cache reloading interval. The cache is not reloaded by default.
cacheReloadTimeBlackList	The periods during which cache reloading is not allowed. This parameter takes effect if the cache parameter is set to ALL. During the periods specified by this parameter, the cache is not reloaded (for example, in Double 11).	No	Optional. This parameter is empty by default. Example: '**2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00**'. • Separate multiple periods with commas (,). • Separate the start time and end time of each period with a hyphen and a greater-than sign (->).
partitionedJoin	Specifies whether to enable Partitioned All Cache.	No	By default, this parameter is set to false, which indicates that Partitioned All Cache is disabled. If Partitioned All Cache is enabled, data is shuffled before the source table is associated with the dimension table based on join keys. • If the cache parameter is set toLRU, the cache hit rate increases. • If the cache parameter is set toALL, memory consumption is reduced because only the required data is cached for each concurrent job.

Sample code

```
 CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id)
) with (
  type = 'odps',
  endPoint = 'your_end_point_name',
  project = 'your_project_name',
  tableName = 'your_table_name',
  accessId = 'your_access_id',
  accessKey = 'your_access_key',
  `partition` = 'ds=20180905',
  cache = 'ALL'
);
```

6.6.7.4. Asynchronous mode

You can enable the asynchronous mode and configure related parameters to improve throughput when you join dimension tables.

Background information

By default, the synchronous mode is used when you join dimension tables. The system queries the physical table and returns the association result each time a data record is added to the physical table. This results in low throughput and high latency. The asynchronous mode is introduced to process query requests in parallel, so that consecutive requests do not need to wait for processing.

Flink SQL implements asynchronous JOIN operations for dimension tables based on Flink Async I/O and asynchronous clients. This significantly improves throughput.

Optimization method

Add **async='true'** to the WITH clause in the data definition language (DDL) statement of a dimension table. The following table describes the parameters related to the asynchronous mode.

Parameter	Description	Required	Remarks
async	Specifies whether to enable the asynchronous mode.	No	Default value: false.
asyncResultOrder	Specifies whether to sort asynchronous results.	No	Valid values: • unordered (default value) • ordered
asyncTimeoutMs	The timeout period of an asynchronous request, in milliseconds.	No	Default value: 180000.
asyncCapacity	The maximum number of asynchronous requests in an asynchronous request queue.	No	Default value: 100.
asyncCallbackThreads	The number of callback threads.	No	An asynchronous request is implemented by a thread. onComplete is called if an asynchronous request succeeds and onError is called if it fails. The number of times onComplete and onError is called determines the size of the thread pool. Default value: 50.

asyncConnectionQueueMaxsize	The maximum number of requests that can be sent.	No	If the number of requests waiting for a server to process reaches the value specified by this parameter, asynchronous request calling is blocked to prevent out of memory of the client. Default value: 100.
asyncCallbackQueueMaxsize	The maximum number of requests in a callback processing queue.	No	If the number of requests waiting for callback processing reaches the value specified by this parameter, asynchronous request calling is blocked to prevent out of memory of the client. Default value: 500.

Sample code

```
CREATE TABLE dim_cn_item(
    rowkey VARCHAR,
    item_id VARCHAR,
    title VARCHAR,
    cate_id VARCHAR,
    cate_name VARCHAR,
    cate_level1_id VARCHAR,
    cate_level2_id VARCHAR,
    cate_level3_id VARCHAR,
    cate_level1_name VARCHAR,
    cate_level2_name VARCHAR,
    cate_level3_name VARCHAR,
    pinlei_id VARCHAR,
    pinlei_name VARCHAR,
    bu_id VARCHAR,
    bu_name VARCHAR,
    PRIMARY KEY(rowkey)
) WITH(
    type='alihbase',
    diamondKey = 'xxxx',
    diamondGroup ='yyyy',
    cacheTTLMs='3600000',
    async='true',
    cache='LRU',
    columnFamily='cf',
    cacheSize='1000',
    tableName='yourTableName'
);
```

FAQ

- Error information

```
Caused by: org.apache.flink.table.api.TableException: Output mode can not be UNORDERED if the input is an update stream.
```

```
at
org.apache.flink.table.plan.util.TemporalJoinUtil$.validate(TemporalJoinUtil.scala:340)

at
org.apache.flink.table.plan.nodes.common.CommonTemporalTableJoin.translateToPlanInterna
CommonTemporalTableJoin.scala:144)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translate
PlanInternal(StreamExecTemporalTableJoin.scala:98)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translate
PlanInternal(StreamExecTemporalTableJoin.scala:39)
at
org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:58

at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.org$apach
flink$table$plan$nodes$exec$StreamExecNode$$super$translateToPlan(StreamExecTemporalTab
Join.scala:39)
at
org.apache.flink.table.plan.nodes.exec.StreamExecNode$class.translateToPlan(StreamExecN
e.scala:38)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translate
Plan(StreamExecTemporalTableJoin.scala:39)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translate
Plan(StreamExecTemporalTableJoin.scala:39)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlanInterna
StreamExecCalc.scala:89)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlanInterna
StreamExecCalc.scala:43)
at
org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:58

at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.org$apache$flink$table
lan$nodes$exec$StreamExecNode$$super$translateToPlan(StreamExecCalc.scala:43)
at
org.apache.flink.table.plan.nodes.exec.StreamExecNode$class.translateToPlan(StreamExecN
e.scala:38)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlan(Stream
ecCalc.scala:43)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translate(StreamExecSi
.scala:158)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translateToPlanInterna
StreamExecSink.scala:103)
at
org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translateToPlanInterna
```

```
StreamExecSink.scala:53)
at
org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:58

at
```

- Cause
  If the upstream data is Update Stream, the asyncResultOrder parameter is set to unordered when you join dimension tables.

  > **Note** Update Stream can be the TopN operation or the JOIN operation on two data streams.

- Solution

  Set the **asyncResultOrder** parameter to ordered in the WITH clause for the dimension table.

6.6.7.5. APPROX_COUNT_DISTINCT

You can optimize your job performance by using the APPROX_COUNT_DISTINCT function. Compared with COUNT(DISTINCT), this function returns an approximate count.

Background information

When the COUNT(DISTINCT) function is used, distinct key information is saved in state data of the aggregate node. If a large number of distinct keys exist, the read/write overhead of state data is large. This causes a bottleneck in job performance optimization. In many cases, accurate computation is not necessary. If you are willing to achieve high job performance at the expense of accuracy, you can use the **APPROX_COUNT_DISTINCT** function. **APPROX_COUNT_DISTINCT** supports miniBatch and local-global optimization on the aggregate node. When you use this function, make sure that the following requirements are met:

- The input data does not contain retracted messages.
- A large number of distinct keys, such as unique visits (UVs), exist. The **APPROX_COUNT_DISTINCT** function cannot bring obvious benefits if only a small number of distinct keys exist.

Optimization method

Use **APPROX_COUNT_DISTINCT(user)** to replace **COUNT(DISTINCT user)** in the SQL. The syntax of **APPROX_COUNT_DISTINCT(user)** is:

```
APPROX_COUNT_DISTINCT(col [, accuracy])
```

where:

- col indicates the name of a field, which can be of any type.
- accuracy specifies the calculation accuracy. A larger value indicates higher accuracy, higher state overhead, and lower performance. This field is optional. Valid values: **(0.0, 1.0)**. Default value: 0.99.

Sample code

- Test data

a (VARCHAR)	c (BIGINT)

Hi	1
Hi	2
Hi	3
Hi	4
Hi	5
Hi	6

- Test statement

```
SELECT
  a,
  APPROX_COUNT_DISTINCT(b) as b,
  APPROX_COUNT_DISTINCT(b, 0.9) as c
FROM MyTable
GROUP BY a;
```

- Test results

a (VARCHAR)	b (BIGINT)	c (BIGINT)
Hi	5	5

6.6.7.6. Local-global optimization

You can use local-global optimization to resolve data skew issues of an aggregate node.

Background information

When you use local-global optimization, the aggregation process is divided into two phases: local aggregation and global aggregation. They are similar to the combine and reduce phases in MapReduce. In the local aggregation phase, the system aggregates data that is locally buffered at the input node into batches and generates an accumulator for each batch of data. In the global aggregation phase, the system merges the accumulators to obtain the global aggregation result.

Local-global optimization can eliminate data skews by using local aggregation and resolve the data hotspot issues in global aggregation. This improves job performance. You can use local-global optimization to improve the performance of common aggregate functions, such as SUM, COUNT, MAX, MIN, and AVG. You can also use it to resolve data hotspot issues that occur when you use these functions.

Optimization method

A user-defined aggregate function (UDAF) must execute the merge method to trigger local-global optimization. For more information about how to execute the merge method, see Sample code.

> **Note** In Blink 2.0 and later versions, local-global optimization is enabled by default. If you want to disable local-global optimization, set **blink.localAgg.enabled** to false in **job parameters**.

Sample code

```
import org.apache.flink.table.functions.AggregateFunction;

public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
    // Define the data structure of the accumulator that stores state data of the COUNT
UDAF.
    public static class CountAccum {
        public long total;
    }

    // Initialize the accumulator of the COUNT UDAF.
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }

    // The getValue method is used to compute the result of the COUNT UDAF based on the
accumulator that stores state data.
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }

    // The accumulate method is used to update the accumulator that is used by the COUN
T UDAF to store state data 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;
         }
    }
}
```

6.6.7.7. ROW_NUMBER OVER WINDOW

You can use the ROW_NUMBER OVER WINDOW function to efficiently deduplicate source data.

Background information

Deduplication aims to obtain top N records. Realtime Compute for Apache Flink supports two deduplication policies:

- Deduplicate Keep FirstRow: retains only the first record under a key. The state data contains only the key information, so the node performance is high after you enable deduplication by using ROW_NUMBER OVER WINDOW.
- Deduplicate Keep LastRow: retains only the last record under a key. This policy slightly outperforms the LAST_VALUE function.

Optimization method

SQL does not have deduplication syntax. Realtime Compute for Apache Flink uses the ROW_NUMBER OVER WINDOW function to deduplicate data.

```
SELECT *
FROM (
   SELECT *,
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]
    ORDER BY timeAttributeCol [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1;
```

Parameter	Description
ROW_NUMBER()	Calculates the row number. It is a window function that contains an OVER clause. The value starts from 1.
PARTITION BY col1[, col2..]	Optional. Specifies partition columns for storing primary keys of duplicate records.
ORDER BY timeAttributeCol [asc|desc])	Specifies the column by which you want to sort data. You must specify a time attribute, which can be proctime or rowtime. You must also specify the sort order, which can be asc (Deduplicate Keep FirstRow) or desc (Deduplicate Keep LastRow). **Note** • If you do not specify the time attribute, proctime is used by default. • If you do not specify the sort order, asc is used by default.
rownum	The current row number. Only **rownum=1** and **rownum<=1** are supported.

When the ROW_NUMBER OVER WINDOW function is executed, two levels of queries are performed:

1. The **ROW_NUMBER()** function is used to sort data records under a key by the time attribute and mark the records with their rankings.

   Note
   - If the time attribute is proctime, Realtime Compute for Apache Flink removes duplicate records based on the time when the records are processed by Realtime Compute for Apache Flink. In this case, the rankings may vary each time.
   - If the time attribute is rowtime, Realtime Compute for Apache Flink removes duplicate records based on the time when the records are written to Realtime Compute for Apache Flink. In this case, the rankings remain unchanged.

2. The data records are sorted by their rankings and only the first or last one is retained.

Sample code

- Deduplicate Keep FirstRow
  In this example, Realtime Compute for Apache Flink removes duplicate data records in table T based on field b, and retains the first data record that is processed by Realtime Compute for Apache Flink.

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b ORDER BY proctime) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

- Deduplicate Keep LastRow
  In this example, Realtime Compute for Apache Flink removes duplicate data records in table T based on the b and d fields, and retains the last record that is written to Realtime Compute for Apache Flink.

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY rowtime DESC) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

6.6.7.8. Partial-final optimization

You can resolve COUNT(DISTINCT) hotspot issues by using partial-final optimization.

Background information

To resolve the COUNT(DISTINCT) hotspot issues, you can change the aggregation process to two-layer aggregation by adding a scattered layer at which data is scattered based on the distinct keys. This is referred to as partial-final optimization. Blink 2.2.0 and later versions support partial-final optimization. Partial-final optimization is suitable for the following scenarios:

- The COUNT(DISTINCT) function has been used but the performance requirement of an aggregate node is not met.
- The aggregate node where the COUNT(DISTINCT) function is executed does not have user-defined aggregate functions (UDAFs).

> **Note** Partial-final optimization divides the aggregation process into two layers, which causes additional network shuffling. Therefore, resources are wasted if the data amount is small.

Optimization method

On the **Parameters** tab of the **Development** page, set **blink.partialAgg.enabled** to true.

After partial-final optimization is enabled, check whether an expand node is included in the topology or whether the aggregation process has two layers.

Sample code

```
blink.partialAgg.enabled=true
```

6.7. Monitoring and alerting

This topic describes the monitoring and alerting process in Realtime Compute for Apache Flink and how to create alert rules in Realtime Compute for Apache Flink.

Introduction to CloudMonitor

CloudMonitor helps you collect the monitoring metrics of cloud resources or other custom monitoring metrics, check service availability, and configure alerts based on these monitoring metrics. CloudMonitor helps you view the cloud resource usage, business information, and service health status. In addition, you can receive alerts and respond to these alerts at the earliest opportunity to keep your applications running properly.

Create alert rules

For more information about how to create an alert rule, see Cloud service monitoring.

Monitoring items of Realtime Compute for Apache Flink

Monitoring item	Unit	Metric	Dimensions	Statistics
Service delay	s	inputDelay	userId, regionId, projectName, and jobName	Average
Read records per second (RPS)	RPS	ParserTpsRate	userId, regionId, projectName, and jobName	Average
Write RPS	RPS	SinkOutTpsRate	userId, regionId, projectName, and jobName	Average
Failover rate **Note** The failover rate is the average number of failovers per second in the last minute. For example, if one failover occurred in the last minute, the failover rate is 0.01667 (1/60 = 0.01667).	%	TaskFailoverRate	userId, regionId, projectName, and jobName	Average

Processing delay	s	FetchedDelay	userId, regionId, projectName, and jobName	Average

View monitoring metrics

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration**.
3. On the **Administration** page, click the name of the job for which you want to view monitoring metrics.
4. In the upper-right corner of the page, choose **More > Monitor**.
5. On the page that appears, view the monitoring metrics of the job.

6.8. Customize log levels and download paths

You can configure the parameters of a Realtime Compute for Apache Flink job to customize the log download levels and download paths.

> **Important** Only Realtime Compute for Apache Flink V3.2 and later allows you to customize the log levels and paths.

Procedure

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. In the **Development** section of the left-side navigation pane, double-click the folder that stores the required job and find the job.
4. Double-click the job to go to the job editing page.
5. On the right side of the job editing page, click the **Parameters** tab. In the pane that appears, enter the configuration data of Log4j.

Log4j configuration parameter	Description

Root logger	Processes some operations for logging. The syntax of the root logger is `log4j.rootLogger = [ level ] , appenderName, appenderName, …` . The syntax indicates that the log records whose level is at least the specified level are delivered to one or more destinations. The following items describe the parameters in the syntax: ◦ **level**: specifies the level of logs. The levels of logs include ERROR, WARN, INFO, and DEBUG that are sorted in descending order. ERROR indicates serious errors for the job. WARN indicates potentially harmful situations for the job. INFO indicates informational messages for the job. DEBUG indicates the debugging information about the job. ◦ **appenderName**: specifies the name of the destination to which log records are delivered. You can specify multiple appenders for the root logger.

Appender	Specifies the destination to which log records are delivered. To define an appender, use the following syntax:
	`log4j.appender.appenderName = fully.qualified.name.of.appender.class`
	`log4j.appender.appenderName.option1 = value1`
	`...`
	`log4j.appender.appenderName.optionN = valueN`
	Appenders have the following two types:
	◦ Appenders provided by Log4j
	▪ `org.apache.log4j.ConsoleAppender` : delivers log records to a console.
	▪ `org.apache.log4j.FileAppender` : delivers log records to a file.
	▪ `org.apache.log4j.DailyRollingFileAppender` : generates a log file every day.
	▪ `org.apache.log4j.RollingFileAppender` : generates a new file when a file reaches its maximum size.
	▪ `org.apache.log4j.WriterAppender` : delivers log records in the stream format to a specified destination.
	◦ Custom appenders
	Custom appenders allow you to deliver log records to only the Log Service and Object Storage Service (OSS) services. The classes of the two services are fixed. The following example lists the fixed classes.
	▪ Log Service: `log4j.appender.loghub=com.alibaba.blink.log.loghub.BlinkLogHubAppender`
	▪ OSS: `log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender`
	You can also specify other configuration items that are supported by the Log4j syntax.
	For more information, see Change the log level to DEBUG and deliver log records to an OSS bucket..

> **Note** After you specify the job parameters, restart the job. You can view the new logs in the OSS bucket.

6. **Terminate** the job. For more information, see Terminate a job.
7. **Start** the job. For more information, see Start a job.

Considerations

- The OSS path must be the same as that for the Realtime Compute for Apache Flink cluster that is deployed in exclusive mode.
- **The logs that you can download** are provided through Log4j. system.out logs are not included in these logs.
- The specified Log Service or OSS can interact with the cluster where the job resides.
- The job can be started in most cases even if the custom log output configuration is invalid. However, the logs of the job cannot be displayed based on the configuration.

Change the log level to DEBUG and deliver log records to an OSS bucket.

> **Important** If you specify the **log4j.rootLogger** parameter, you may fail to view log information or troubleshoot related issues on the Realtime Compute for Apache Flink development platform. Use this parameter with caution.

```
#Change the log level to DEBUG and export logs to a specific file in an OSS bucket.
log4j.rootLogger=DEBUG, file, oss

#This parameter setting is fixed. You do not need to change the setting. Configure the
appender class for OSS.
log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender

#The endpoint.
log4j.appender.oss.endpoint=oss-cn-hangzhou****.aliyuncs.com

#The AccessKey ID.
log4j.appender.oss.accessId=U****4ZF

#The AccessKey secret.
log4j.appender.oss.accessKey=hsf****DeLw

#The OSS bucket name.
log4j.appender.oss.bucket=et****

#The subdirectory that is used to store logs.
log4j.appender.oss.subdir=/luk****/test/
```

Disable the output of the logs for a specified package, and deliver logs to a specified Logstore of Log Service.

```
#Disable the output of the logs for the log4j.logger.org.apache.hadoop package.
log4j.logger.org.apache.hadoop = OFF

#This parameter setting is fixed. You do not need to change the setting. Configure the
LogHub appender.
log4j.appender.loghub = com.alibaba.blink.log.loghub.BlinkLogHubAppender

#Deliver only logs of the ERROR level to Log Service.
log4j.appender.loghub.Threshold = ERROR

#The name of a project in Log Service.
log4j.appender.loghub.projectName = blink-errdumpsls-test

#The name of a Logstore in Log Service.
log4j.appender.loghub.logstore = logstore-3

#The endpoint of Log Service.
log4j.appender.loghub.endpoint = http://cn-shanghai****.sls.aliyuncs.com

#The AccessKey ID.
log4j.appender.loghub.accessKeyId = Tq****WR

#The AccessKey secret.
log4j.appender.loghub.accessKey = MJ****nfVx
```

Change the log level to WARN, and disable the output of the logs for a specified package.

```
#Change the log level to WARN.
log4j.rootLogger=WARN,file

#Disable the output of the logs for the log4j.logger.org.apache.hadoop package.
log4j.logger.org.apache.hadoop = OFF
```

6.9. Manage Blink versions of a Realtime Compute for Apache Flink cluster deployed in exclusive mode

Realtime Compute for Apache Flink provides the version management feature for you to manage Blink versions of a Realtime Compute for Apache Flink cluster in exclusive mode.

Install a version

1. Go to the Version Management page
   i. Log on to the Realtime Compute development platform.

ii. Move the pointer over the username in the top navigation bar and click **Project Management**.

iii. In the left-side navigation pane, choose **Cluster Management > Clusters**.

iv. On the **Clusters** page, find the cluster for which you want to manage Blink versions. In the **Actions** column, choose **More > Version Management**.

2. On the **Installable Version** tab, find the version you want to install, and click **Install** in the **Actions** column.
On the **Installable Version** tab, choose a proper version based on your business requirements and the version features.

Tag	Description
stable	The recommended stable Blink version.
beta	The beta version programmed for testing. **Note** We recommend that you install a beta version in only specified scenarios. The performance and the stability of the beta version cannot be ensured in other scenarios.
No tag	The historical stable version.

Note
- You can install one version at a time. During installation, the cluster status is **Version Installing**.
- You cannot install a version that has been installed.
- Each version is configured with a service period. The default service period is one year. After the service period expires, you can continue to use the version, but Alibaba Cloud no longer maintains the version.

Switch a version

Blink features vary based on Blink versions. You can configure a Blink version based on your business requirements. You can **switch the Blink version** for a job.

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Development**.
3. On the **Development** page, double-click the job in a folder that stores the job to go to the **job editing** page.
4. In the right side of the **job editing** page, click Version Information.
5. From the version list, select the Blink version you want to switch to.

Uninstall a Blink version

Realtime Compute for Apache Flink allows you to install only three Blink versions. If you already install three Blink versions and need to install a new version, you must uninstall an existing version.

1. Go to the Version Management page

i. Log on to the Realtime Compute development platform.

ii. Move the pointer over the username in the top navigation bar and click **Project Management**.

iii. In the left-side navigation pane, choose **Cluster Management > Clusters**.

iv. On the **Clusters** page, find the cluster for which you want to uninstall a version. In the **Actions** column, choose **More > Version Management**.

2. On the **Installed Versions** tab, find the version, and click **Uninstall** in the **Actions** column.

> **Note**
> - You cannot uninstall a Blink version that is set to **current**.
> - You cannot uninstall a Blink version that is referenced by a Realtime Compute for Apache Flink job.

Specify the current version

You can select an installed Blink version and set the version to the current version of a job in the current cluster. Blink features vary based on Blink versions. You can set a Blink version to the current version of a Realtime Compute for Apache Flink cluster based on your business requirements.

1. Go to the Version Management page
   i. Log on to the Realtime Compute development platform.
   ii. Move the pointer over the username in the top navigation bar and click **Project Management**.
   iii. In the left-side navigation pane, choose **Cluster Management > Clusters**.
   iv. On the **Clusters** page, find the cluster for which you want to set a version to the current version. In the **Actions** column, and choose **More > Version Management**.
2. On the **Installed Version** page, find the version that you want to set to the current version, and click **Set to current** in the **Actions** column.

FAQ

- What can I do if the `blink-<version> already installed` error message appears during installation?

  The error message is returned because the Blink version has been installed. You do not need to install the version again.

- What can I do if the `Flink versions exceeded max limitation:3` error message appears during installation?

  The error message is returned because the number of versions exceeds the upper limit. To install a new version, you must delete existing versions until the number of installed versions is less than three.

- What can I do if the `Node:<nodeName> in project:<projectName> still ref the version:<blinkVersion>` error message appears during uninstallation?

  The error message is returned because the Blink version is referenced by an online job. You must bring the job offline based on the job name and project name provided in the error message.

- What can I do if the following message appears when I click **Syntax Check** or **Publish**?

```
code:[30006], brief info:[blink script not exist, please check blink version], context info:[blink script:[/home/admin/blink/blink-2.2.6-hotfix0/bin/flink], blink version:[/home/admin/blink/blink-2.2.6-hotfix0/bin/flink]]
```

The error message is returned because the Blink version used by the job does not exist. In the Realtime Compute for Apache Flink console, choose **Version Management > Installed Versions** to check whether the version exists,If the version does not exist, switch to another version or install the version.

6.10. Monitoring and alerting

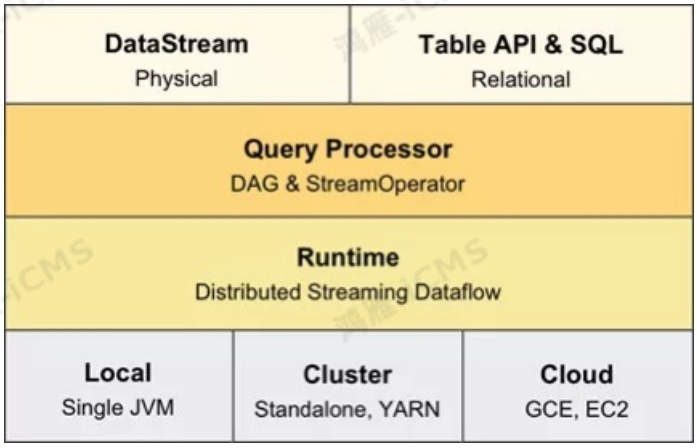
7.Blink Datastream Development Guide

7.1. Overview

Alibaba Cloud Realtime Compute for Apache Flink allows you to develop, publish, and start Flink DataStream jobs. It also supports O&M management as well as monitoring and alerting features.

> **Important**
> - Only Blink 3.2.2 and later versions of Realtime Compute for Apache Flink in exclusive mode supports Flink DataStream.
> - Flink DataStream does not support storage resource registration, job debugging, or configuration optimization for Flink SQL jobs.
> - If the upstream and downstream storage systems that Flink DataStream jobs access use a whitelist mechanism, you must configure whitelists. For more information, see Configure a whitelist for accessing storage resources.
> - Flink DataStream jobs supported by Realtime Compute for Apache Flink are developed based on the open source Flink version. For more information, see Open source Flink version.
> - Blink DataStream APIs are compatible with open source Flink 1.5. Flink DataStream jobs and connectors that are developed based on Flink 1.5 can run properly on Blink. Blink DataStream APIs may not be compatible with other open source Flink versions.

Flink DataStream enables Realtime Compute for Apache Flink to call underlying API operations to help you flexibly use Realtime Compute for Apache Flink.



Flink DataStream Developer Guide includes the following topics:

- Job development

  This topic describes how to develop, publish, and start a Flink DataStream job in Realtime Compute for Apache Flink.

- Monitoring and alerting

  This topic describes how to create and apply alert rules. Monitoring and alerting are supported only for the failover rate metric.

7.2. Configure a whitelist for accessing storage resources

By default, a newly created database instance does not allow access from IP addresses that are not included in its whitelists. To allow Realtime Compute for Apache Flink to access the database instance, you must add the IP addresses of Realtime Compute for Apache Flink to a whitelist of the database instance. This topic describes how to add the IP addresses of Realtime Compute for Apache Flink to a whitelist of an ApsaraDB for RDS instance.

IP addresses to be added to the whitelist

To access storage resources from a Realtime Compute for Apache Flink cluster in exclusive mode, you only need to add the IP addresses of the ENI to the whitelist. To view the IP addresses of the ENI, perform the following steps:

1. Log on to the .
2. Move the pointer over the username in the upper-right corner.
3. In the drop-down list, click **Project Management**.
4. In the left-side navigation pane, click **Clusters**.
5. On the **Clusters** page, click the name of the target cluster.
6. In the cluster information dialog box, view the **ENI** of the cluster.

Configure a whitelist for an ApsaraDB for RDS instance

When you reference an ApsaraDB for RDS database in Realtime Compute for Apache Flink, Realtime Compute for Apache Flink needs to frequently read and write data in the ApsaraDB for RDS database. In this case, you must add the IP addresses of Realtime Compute for Apache Flink to a whitelist of the ApsaraDB for RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for MySQL instance.

7.3. Set custom parameters

In a DataStream job, you can obtain custom parameters from the main function as needed.

Method to obtain and set custom parameters

To obtain a custom parameter from the main function, define the custom parameter in the format of `paramName=paramValue` in a DataStream job.

- `paramName`: the name of the custom parameter.
- `paramValue`: the value of the custom parameter.

> **Note** You can define multiple custom parameters in a DataStream job.

Example of obtaining and setting custom parameters

> **Note** After you set `blink.job.name=jobnametest` on the development page, you can use the following code to assign the `jobnametest` string to the `jobName` variable.

```
import org.apache.flink.api.java.utils.ParameterTool;
import java.io.StringReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Properties;

public class getParameterExample {
    public static void main(String[] args)throws Exception {
        final String jobName;
        final ParameterTool params = ParameterTool.fromArgs(args);

        /*You must enter configFile to read parameters in configFile. */
        String configFilePath = params.get("configFile");

        /*Create a Properties object to save parameter values that you set in the system. */
        Properties properties = new Properties();

        /*Load parameter values that you set in the system to the Properties object. */
        properties.load(new StringReader(new String(Files.readAllBytes(Paths.get(configFilePath)), StandardCharsets.UTF_8)));

        /*Obtain the jobName parameter. */
        jobName = (String) properties.get("blink.job.name");
    }
}
```

Note The preceding example only demonstrates how to obtain and use custom parameters in the main function. To use custom parameters in a Flink operator, you must add the following code to the preceding code to convert custom parameters to global job parameters. Then, you can obtain the corresponding parameters in a Flink operator by using the getproperty method of the Configuration class.

```
env.getConfig().setGlobalJobParameters(ParameterTool.fromPropertiesFile(configFilePath));
```

7.4. Monitoring

This topic describes monitoring and alerts in Realtime Compute for Apache Flink and how to create an alert rule.

Introduction to CloudMonitor

CloudMonitor helps you collect the monitoring metrics of Alibaba Cloud resources or custom monitoring metrics, check service availability, and configure alert rules based on these monitoring metrics. This service helps you view Alibaba Cloud resource usage, business running information, and service health status. In addition, this service allows you to receive and respond to alerts in a timely manner to ensure that applications are running as expected.

View monitoring metrics

1. Log on to the Realtime Compute for Apache Flink console.

2. In the top navigation bar, click **Administration**.
3. On the **Administration** page, click the name of the job whose monitoring metrics you want to view.
4. In the upper-right corner of the Job Administration page, move the pointer over the **More** icon and click **Monitor**.
5. On the page that appears, view the monitoring metrics of the job.

Create an alert rule

For more information about how to create an alert rule, see Cloud service monitoring.

> **Note**
>
> - The failover rate is the average number of failovers per second in the last minute. For example, if one failover occurred in the last minute, the failover rate is 0.01667 (1/60 = 0.01667).
> - If you use a connector provided by the open source Flink during the development of a Flink DataStream job, the monitoring metrics **service latency**, **read RPS**, and **write RPS** are not displayed in the CloudMonitor console.

7.5. Develop a job

This topic describes the POM dependency used to develop DataStream jobs, example of DataStream job development, and DataStream connectors.

> **Important**
>
> - Only Blink 3.2.2 and later versions of Realtime Compute for Apache Flink in exclusive mode supports Flink DataStream.
> - We recommend that you use a Maven project of IntelliJ IDEA to develop a DataStream job.
> - To avoid JAR dependency conflicts, take note of the following points:
>   - Select the Blink version on the Development page the same as the Blink version of POM dependencies.
>   - Specify `<scope>provided</scope>` for Blink-related dependencies.
>   - Use the Shade plug-in to package other third-party dependencies. For more information, see Apache Maven Shade plug-in.

POM dependency

Add a POM dependency based on the Blink version of the job that is running. The following example shows the POM file used for Blink 3.4.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-datastreaming</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <properties>
        <scala.version>2.11.12</scala.version>
        <scala.binary.version>2.11</scala.binary.version>
        <blink.version>blink-3.4.0</blink.version>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.alibaba.blink</groupId>
            <artifactId>flink-streaming-java_${scala.binary.version}</artifactId>
            <version>${blink.version}</version>
            <scope>provided</scope>
        </dependency>

<!--        Add test framework-->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.scala-lang</groupId>
            <artifactId>scala-library</artifactId>
            <version>2.11.12</version>
        </dependency>

<!--        Add logging framework-->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.7</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>



    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.0</version>
                <executions>
                    <execution>
```

```
                    <execution>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <manifestEntries>
                                        <Main-Class>your main class</Main-Class>
                                        <X-Compile-Source-JDK>${maven.compiler.source}</X
-Compile-Source-JDK>
                                        <X-Compile-Target-JDK>${maven.compiler.target}</X
-Compile-Target-JDK>
                                    </manifestEntries>
                                </transformer>
                            </transformers>
                            <relocations combine.self="override">
                                <relocation>
                                    <pattern>XXX</pattern>
                                    <shadedPattern>shaded.XXX</shadedPattern>
                                </relocation>
                            </relocations>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

> **Note** If you want to use Snapshot, you can add the POM dependency based on your Snapshot version.

DataStream connectors

The following DataStream connectors are added to Blink 3.2:

- Kafka
- Kafka (open source version)
- HBase (open source version)
- JDBC
- RDS SINK
- Elasticsearch
- MongoDB
- Redis

> **Note** Some DataStream connectors have been open-sourced. For more information, see alibaba-flink-connectors.

7.6. Publish a job

This topic describes how to publish a DataStream job.

Prerequisites

A Realtime Compute for Apache Flink project is created.

> **Important**
> - Only Blink 3.2.2 and later of Realtime Compute for Apache Flink in exclusive mode supports Flink DataStream. We recommend that you use Blink 3.4.0 or later.
> - DataStream jobs do not support resource configuration optimization or the setting of a start offset. For Blink versions earlier than 3.4.0, we recommend that you use the default configurations when you publish and start a job.

Procedure

1. Log on to the Realtime Compute for Apache Flink console. In the top navigation bar, click **Development**.
2. On the **Development** page, click the **Create File** icon.
3. In the **Create File** dialog box, configure job parameters.

Parameter	Description
File Name	The name of the custom job. The file name must be unique in the current project.
File Type	The type of the job. Set it to FLINK_STREAM / DATASTREAM. **Note** You must set File Type to `FLINK_STREAM / DATASTREAM` for both DataStream jobs and Table API jobs.
Storage Path	The path where the job is stored.

4. In the left-side navigation pane, click the **Resources** tab.
5. On the Resources tab, click **Create Resource**. In the dialog box that appears, configure required parameters, upload the JAR package of the developed DataStream job, and then click OK.

> **Note** The maximum size of the JAR package that can be uploaded is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use APIs to upload it.

6. In the left-side navigation pane, find the uploaded package, click More in the Actions column, and click **Reference**.
7. On the **Development** page for the job, configure required parameters.

```
blink.main.class=<Complete main class name>
-- The complete function class name, for example,
com.alibaba.realtimecompute.DemoTableAPI.
blink.job.name=<Job name>
-- For example, datastream_test.
blink.main.jar=<Resource name of the JAR package of the complete main class name>
-- Resource name of the JAR package of the complete main class name, for example, bli
nk_datastream.jar.
```

- **blink.main.class** and **blink.job.name** are required. Make sure that the value of **blink.job.name** is the same as the file name you entered in Step 3. If they are different, the file name you entered in Step 3 is used.
- You must set the **blink.main.jar** parameter when you upload multiple JAR packages.
- You can configure other parameters as required and then reference them in Realtime Compute for Apache Flink. For more information about how to configure custom parameters and how to obtain parameter values from the code.
- Do not use spaces when you configure parameters.
- In Blink 3.2.0 and later versions, you do not need to set the directory where the checkpoint file is stored. The system automatically generates the directory.
- In Blink 3.4.0 and later versions, the parameter configurations in the code of the JAR package take precedence over the parameter configurations in Realtime Compute for Apache Flink. For example:
  - If the **statebackend** parameter is configured both in custom parameters and the code of the JAR package, the configuration of this parameter in the code of the JAR package is used.
  - If the **statebackend** parameter is not configured in custom parameters and the code of the JAR package, the default parameter **niagara statebackend** in the job template of the Realtime Compute for Apache Flink development platform is used.

    > **Note** Exercise caution when you delete default parameters from the job template. If you delete the default parameters, checkpoint generation and fault tolerance for the job may fail. The **blink.job.name** parameter is an exception. The job name configured in **env.execute("jobname")** in the code will be replaced with the job name that you configured when you create the job. This ensures that the value of the blink.job.name parameter is consistent with the job name you configured. The job names in metrics that include custom metrics must also be the same as the job names you configured when you create the jobs.

8. Publish the job.
   - Blink versions earlier than 3.4.0
     a. Configure resources.
     Specify the resource configuration mode as required. We recommend that you use the default configuration if you start the job for the first time.

     > **Note** Realtime Compute for Apache Flink supports manual resource configuration. For more information, see Optimize performance by manual configuration.

     b. Check data.

     Check parameter settings and click **Next**.

     c. Publish the job.

     Click **Publish**.
   - Blink 3.4.0 and later
     a. Click **Publish**.

b. Specify **Resource Configuration Method**.
   - **Code Configuration**: uses the resource configurations in the code. This resource configuration method is consistent with that used by the open source Flink.
   - **Manual**: uses the resource configurations that are manually adjusted on the **Resource Configuration** page.
     a. In the right side of the **Development** page, click the **Configurations** tab. Then, choose **Configurations > Reacquire Configuration**.
     b. Modify the configurations as required.
     c. Choose **Configurations > Apply** to save the configurations.

     > **Note** During manual configuration, the resource configurations in the code take precedence over those displayed on the Realtime Compute for Apache Flink development platform. For example, if the resources of some operators are configured in the code, the configurations of the resources of these operators on the Realtime Compute for Apache Flink development platform become invalid. During job running, the resource configurations of the operators in the code are used. For resource configurations that are not displayed in the code, the configurations displayed on the Realtime Compute for Apache Flink development platform are used.

c. Click **Next** to check the setting or click **Skip Check**.
d. Click **Publish**.

9. On the **Administration** page, find the target job and click **Start** in the **Actions** column.

7.7. Develop a job

7.8. Example of DataStream jobs

7.8.1. Read data from DataHub

This topic describes how to run a Flink DataStream job to read data from DataHub.

Prerequisites

- Java Development Kit (JDK) 8 is installed on your on-premises machine.
- Maven 3.x is installed on your on-premises machine.
- An integrated development environment (IDE) for Java or Scala is installed on your on-premises machine. We recommend that you use IntelliJ IDEA. The JDK and Maven are configured.
- A topic is created in DataHub, and test data exists in the topic.

  > **Note** The test data must contain four fields, whose data types are STRING, STRING, DOUBLE, and BIGINT in sequence.

- datahub-demo-master is downloaded.

Background information

Windows OS and macOS are used in this demo.

> **Important** Only Blink 3.X supports this demo.

Develop a job

1. DataStream of Realtime Compute for Apache Flink is compatible with Apache Flink 1.5.2. Download and decompress flink-1.5.2-compatible to your on-premises machine.

   > **Note** **datahub-connector** in the downloaded file functions as the DataHub sink. For more information, see **DatahubSinkFunction.java** and **DatahubSinkFunctionExample.java** in the downloaded file.

2. In the command window, go to the **alibaba-flink-connectors-flink-1.5.2-compatible** directory and run the following command:

   ```
   mvn clean install
   ```

   The following figure shows the command results.

   ```
   [INFO] ------------------------------------------------------------------------
   [INFO] Reactor Summary for aliyun-flink-connectors-parent 0.1-SNAPSHOT:
   [INFO]
   [INFO] aliyun-flink-connectors-parent ..................... SUCCESS [04:54 min]
   [INFO] aliyun-connectors-common ........................... SUCCESS [01:55 min]
   [INFO] cloudhbase-connector ............................... SUCCESS [04:47 min]
   [INFO] datahub-connector .................................. SUCCESS [ 24.694 s]
   [INFO] sls-shaded-sdk ..................................... SUCCESS [ 23.142 s]
   [INFO] sls-connector ...................................... SUCCESS [ 10.359 s]
   [INFO] ------------------------------------------------------------------------
   [INFO] BUILD SUCCESS
   [INFO] ------------------------------------------------------------------------
   [INFO] Total time:  12:39 min
   [INFO] Finished at: 2020-03-10T18:20:03+08:00
   [INFO] ------------------------------------------------------------------------
   ```

   If the command is successfully run, the Java Archive (JAR) package that corresponds to **datahub-connector** is installed in the Maven repository on your on-premises machine. By default, the package is saved in the **.m2** folder under the folder of the current logon user.

3. Run the following command to check whether the package **datahub-connector-0.1-SNAPSHOT-jar-with-dependencies.jar** exists. This package contains a JAR package and its dependent third-party JAR packages, which will be used in subsequent operations.

   - Windows OS

     ```
     dir C:\Users\Username\.m2\repository\com\alibaba\flink\datahub-connector\0.1-SNAPSHOT
     ```

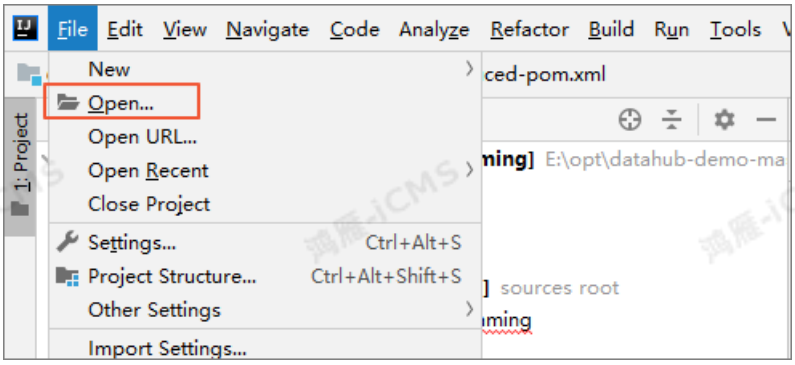
     Figure 1. Command results in the Windows OS

     ```
     C:\Users\[redacted]\.m2\repository\com\alibaba\flink\datahub-connector\0.1-SNAPSHOT [redacted]

     2020/03/10  18:19    <DIR>          .
     2020/03/10  18:19    <DIR>          ..
     2020/03/10  18:19         8,341,200 datahub-connector-0.1-SNAPSHOT-jar-with-dependencies.jar
     2020/03/10  18:19            39,854 datahub-connector-0.1-SNAPSHOT.jar
     2019/09/29  10:37             6,897 datahub-connector-0.1-SNAPSHOT.pom
     2020/03/10  18:19               931 maven-metadata-local.xml
     2020/03/10  18:19               278 _remote.repositories
     ```

   - macOS

     ```
     ls /Users/Username/.m2/repository/com/alibaba/flink/datahub-connector/0.1-SNAPSHOT
     ```

4. In IntelliJ IDEA, choose **File > Open** to open the decompressed package datahub-demo-master. Then, double-click **pom.xml** to view the code.



> **Important**
> - When you perform local debugging for IDE, you must comment out **<scope>provided</scope>**.
> - In this example, **<classifier>jar-with-dependencies</classifier>** in **datahub-connector-0.1-SNAPSHOT-jar-with-dependencies.jar** in Step 3 is used by default.

5. Modify DataHub-related parameters in the **DatahubDemo.java** file.

```
private static String endPoint = "inner endpoint";// Indicates access over an interna
l network.
//private static String endPoint ="public endpoint";// Indicates access over the Inte
rnet. If you have entered an internal endpoint, you do not need to enter the public e
ndpoint.
private static String projectName = "yourProject";
private static String topicSourceName = "yourTopic";
private static String accessId = "yourAK";
private static String accessKey = "yourAS";
private static Long datahubStartInMs = 0L;// Set the time that corresponds to the sta
rt offset.
```

6. Go to the directory where the **pom.xml** file is saved and run the following command to package the file:

```
mvn clean package
```

A JAR package named **blink-datastreaming-1.0-SNAPSHOT.jar** appears in the destination directory, based on the **artifactId** parameter that you configured in the pom.xml file for your project. This indicates that job development is complete.

Publish a job

For more information about how to publish a job, see Publish a job.

> **Important** Before you publish a job, set the Parallelism parameter for the source table on the **Configurations** tab of the **Development** page. The parallelism setting of the source table cannot be greater than the number of shards in the source table. Otherwise, a JobManager error occurs after the job starts.

The following example shows the job content:

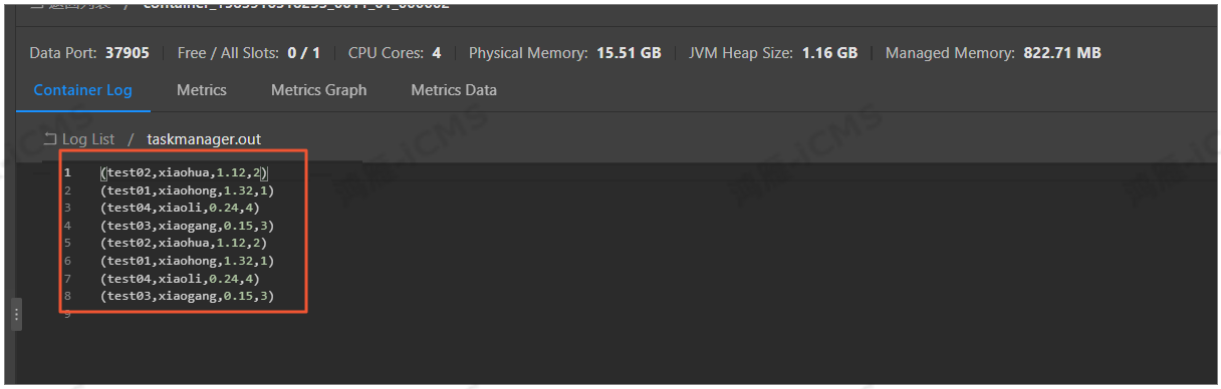
```
-- The complete main class name, such as com.alibaba.realtimecompute.DatastreamExample. This field is required.
blink.main.class=com.alibaba.blink.datastreaming.DatahubDemo

-- The name of the job.
blink.job.name=datahub_demo

-- The resource name of the JAR package that contains the complete main class name, such as blink_datastream.jar.
blink.main.jar=${Resource name of the JAR package that contains the complete main class name}

-- The default state backend configuration. This field takes effect when the job code is not explicitly configured.
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000

-- The default checkpoint configuration. The configuration takes effect when the job code is not explicitly configured.
blink.checkpoint.interval.ms=180000
```

> **Note**
> - Modify **blink.main.class** and **blink.job.name** as required.
> - You can configure custom parameters. For more information, see Set custom parameters.

Verify the test results

On the Container Log tab of the **Job Administration** page, view information in the **taskmanager.out** file of the sink node. In this example, the type of the sink node is print.

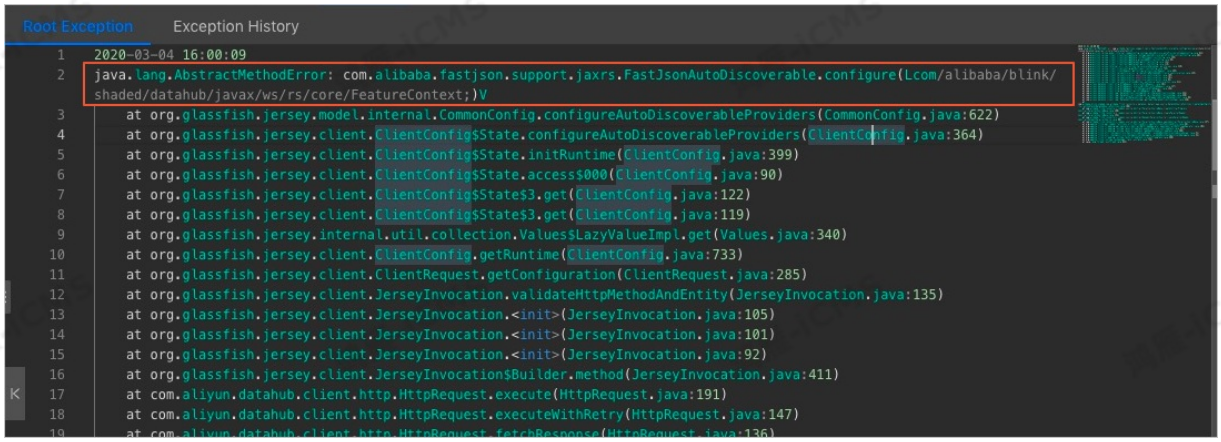
If the information shown in the following figure appears, Realtime Compute for Apache Flink has read data from DataHub.



FAQ

If an error similar to the following error appears when a job is running, a JAR package conflict occurs. What do I do?

```
java.lang.AbstractMethodError:
com.alibaba.fastjson.support.jaxrs.FastJsonAutoDiscoverable.configure(Lcom/alibaba/blink/shaded/datahub/javax/ws/rs/core/FeatureContext;)
```



We recommend that you use the relocation feature of **maven-shade-plugin** to resolve the JAR package conflict.

```
<relocations combine.self="override">
   <relocation>
       <pattern>org.glassfish.jersey</pattern>

<shadedPattern>com.alibaba.blink.shaded.datahub.org.glassfish.jersey</shadedPattern>
   </relocation>
</relocations>
```

7.8.2. Read data from Message Queue for Apache Kafka

This topic describes how to run a Flink DataStream job to read data from Message Queue for Apache Kafka.

Prerequisites

- Java Development Kit (JDK) 8 is installed on your on-premises machine.
- Maven 3.x is installed on your on-premises machine.
- An integrated development environment (IDE) for Java or Scala is installed on your on-premises machine. We recommend that you use IntelliJ IDEA. The JDK and Maven are configured.
- A Message Queue for Apache Kafka instance that resides in the same virtual private cloud (VPC) as your Realtime Compute for Apache Flink cluster in exclusive mode is created. A topic and a consumer group are created.

Background information

- DataStream of Realtime Compute for Apache Flink is compatible with Apache Kafka 1.5.2. Message Queue for Apache Kafka is compatible with Apache Kafka. Therefore, you can directly use the Kafka connector in the Maven repository to access Message Queue for Apache Kafka.
- Realtime Compute for Apache Flink in exclusive mode accesses Message Queue for Apache

Kafka over a VPC. Simple Authentication and Security Layer (SASL) authentication is not required. If you access Message Queue for Apache Kafka over the Internet in your IDE, SASL authentication is required. For more information about the configurations of Message Queue for Apache Kafka, see kafka-java-demo.

> **Important** Only Blink 3.X supports this demo.

Develop a job

1. Download and decompress alikafka-demo-master to your on-premises machine.
2. In IntelliJ IDEA, choose **File > Open** to open the decompressed package alikafka-demo-master.
3. Double-click **kafka.properties** under the \alikafka-demo-master\src\main\resources directory to open the file. Then, change the values of the parameters **bootstrap.servers**, **topic**, and **group.id** to the values of the created Message Queue for Apache Kafka instance.

```
## Endpoints, which are obtained from the Message Queue for Apache Kafka console.

## You can enter public and VPC endpoints for the bootstrap.servers parameter. However, if you use Realtime Compute for Apache Flink in exclusive mode, you must enter VPC endpoints.
bootstrap.servers=ip1:port,ip2:port,ip3:port

## The topic, which is created in the Message Queue for Apache Kafka console.
topic=your_topic

## The consumer group, which is created in the Message Queue for Apache Kafka console.
group.id=your_groupid
```

4. Go to the directory where the **pom.xml** file is stored. Then, run the following command to package the file:

```
mvn clean package
```

A Java Archive (JAR) package named **blink-datastreaming-1.0-SNAPSHOT.jar** appears in the target directory, based on the artifactId parameter that you configured in the pom.xml file for your project. This indicates that job development is complete.

Publish a job

For more information about how to publish a job, see Publish a job.

> **Note** Modify the configurations of **blink.main.class**, **blink.job.name**, and **blink.main.jar** as required.

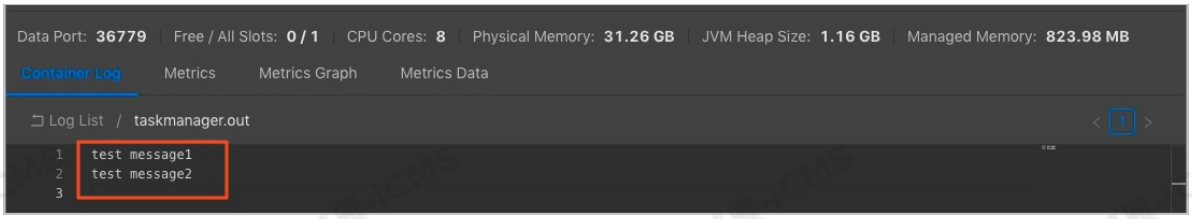
The following example shows the job content:

```
-- The complete main class name, such as com.alibaba.realtimecompute.DatastreamExample.
This field is required.
blink.main.class=com.alibaba.blink.datastreaming.AliKafkaConsumerDemo

-- The name of the job.
blink.job.name=alikafkaconsumerdemo

-- The resource name of the JAR package that contains the complete main class name, suc
h as blink_datastream.jar.
blink.main.jar=blink-datastreaming-1.0-snapshot.jar

-- The default state backend configuration. This field takes effect when the job code i
s not explicitly configured.
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000

-- The default checkpoint configuration. The configuration takes effect when the job co
de is not explicitly configured.
blink.checkpoint.interval.ms=180000
```

> **Note** You can configure custom parameters. For more information, see Set custom parameters.

Verify the test results

1. Send messages to Realtime Compute for Apache Flink in the Message Queue for Apache Kafka console.
2. On the **Job Administration** page, view information in the **taskmanager.out** file of the sink node. In this example, the type of the sink node is print.
   If information similar to that shown in the following figure appears, Realtime Compute for Apache Flink has read data from Message Queue for Apache Kafka. The information depends on the messages sent from the Message Queue for Apache Kafka console.



7.8.3. Read data from DataHub and write data to ApsaraDB for HBase

This topic describes how to run a Flink DataStream job to read data from DataHub and write data to ApsaraDB for HBase.

Prerequisites

- Java Development Kit (JDK) 8 is installed on your on-premises machine.
- Maven 3.x is installed on your on-premises machine.

- An integrated development environment (IDE) for Java or Scala is installed on your on-premises machine. We recommend that you use IntelliJ IDEA. The JDK and Maven are configured.
- A topic is created in DataHub, and test data exists in the topic.

  > **Note**
  >
  > The test data must contain three fields, whose data types are BOOLEAN, STRING, and STRING in sequence.

- An ApsaraDB for HBase cluster is created. The ApsaraDB for HBase cluster resides in the same region and the same virtual private cloud (VPC) as your Realtime Compute for Apache Flink cluster in exclusive mode. A table with several column families is created in the ApsaraDB for HBase cluster. To use Shell to access ApsaraDB for HBase,see Use HBase Shell to access ApsaraDB for HBase Standard Edition clusters.

  > **Note**
  >
  > - ApsaraDB for HBase Standard Edition is used in this topic.
  > - You must add the IP address of your Realtime Compute for Apache Flink cluster to a whitelist of ApsaraDB for HBase.

Background information

The Windows OS is used in this demo.

> **Important**
>
> Only Blink 3.X supports this demo.

Develop a job

1. Download and decompress the Hbase_Demo-master.zip package to your on-premises machine.
2. In IntelliJ IDEA, choose **File > Open** to open the decompressed package Hbase_Demo-master.
3. Double-click the **HbaseDemo.java** file in the \Hbase_Demo-master\src\main\java directory. Then, configure the parameters related to DataHub and ApsaraDB for HBase in the HbaseDemo.java file.

```
// Configure the parameters related to DataHub.
//private static String endPoint ="public endpoint";// Access DataHub over the Intern
et. If you enter an internal endpoint, you do not need to enter the public endpoint.
private static String endPoint = "inner endpoint";// Access DataHub over an internal
network.
private static String projectName = "yourProject";
private static String topicSourceName = "yourTopic";
private static String accessId = "yourAK";
private static String accessKey = "yourAS";
private static Long datahubStartInMs = 0L;// Set the time that corresponds to the sta
rt offset.
// Configure the parameters related to ApsaraDB for HBase.
private static String zkQuorum = "yourZK";
private static String tableName = "yourTable";
private static String columnFamily = "yourcolumnFamily";
```

4. Go to the directory where the **pom.xml** file is stored. Then, run the following command to package the file:

   ```
   mvn package -Dcheckstyle.skip
   ```

   A Java Archive (JAR) package named **Hbase_Demo-1.0-SNAPSHOT-shaded.jar** appears in the target directory, based on the artifactId parameter that you configured in the pom.xml file for your project. This indicates that job development is complete.

Publish a job

For more information about how to publish a job, see Publish a job.

> **Note**
>
> Before you publish the job, set the Parallelism parameter for the source table on the **Configurations** tab of the **Development** page. The parallelism setting of the source table cannot be greater than the number of shards in the source table. Otherwise, a JobManager error occurs after the job starts.

The following example shows the job content:

```
-- Required. The full name of the main class.
blink.main.class=Hbase_Demo.HbaseDemo

-- The name of the job.
blink.job.name=datahub_demo

-- The resource name of the JAR package that contains the full name of the main class.
If multiple JAR packages exist, you must specify this parameter.
--blink.main.jar=Hbase_Demo-1.0-snapshot.jar

-- The default state backend configuration. This field takes effect when the job code i
s not explicitly configured.
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000

-- The default checkpoint configuration. The configuration takes effect when the job co
de is not explicitly configured.
blink.checkpoint.interval.ms=180000
```

> **Note**
> You can configure custom parameters. For more information, see Set custom parameters.

Verify the test results

1. Send test data to DataHub in the Realtime Compute for Apache Flink console.

```
CREATE TABLE kafka_src (
   a BOOLEAN
) WITH (
   type = 'random'
);

CREATE TABLE event_logs (
   `a` BOOLEAN,
    b  VARCHAR,
   `c` VARCHAR
) WITH (
    type = 'datahub',
    endPoint = '<yourEndpoint>',
    project = '<yourProject>',
    topic = '<yourTopic>',
    accessId='<yourAccessId>',
    accessKey='<yourAccessKey>'
);

INSERT INTO event_logs
SELECT
    a,'rowkey3' as b,'123' as c
FROM kafka_src;
```

2. Connect to the ApsaraDB for HBase cluster.For more information about how to connect to an ApsaraDB for HBase cluster, see Use HBase Shell to access ApsaraDB for HBase Standard Edition clusters.
3. Run the `scan 'hbase_sink'` command to query the data written to ApsaraDB for HBase.

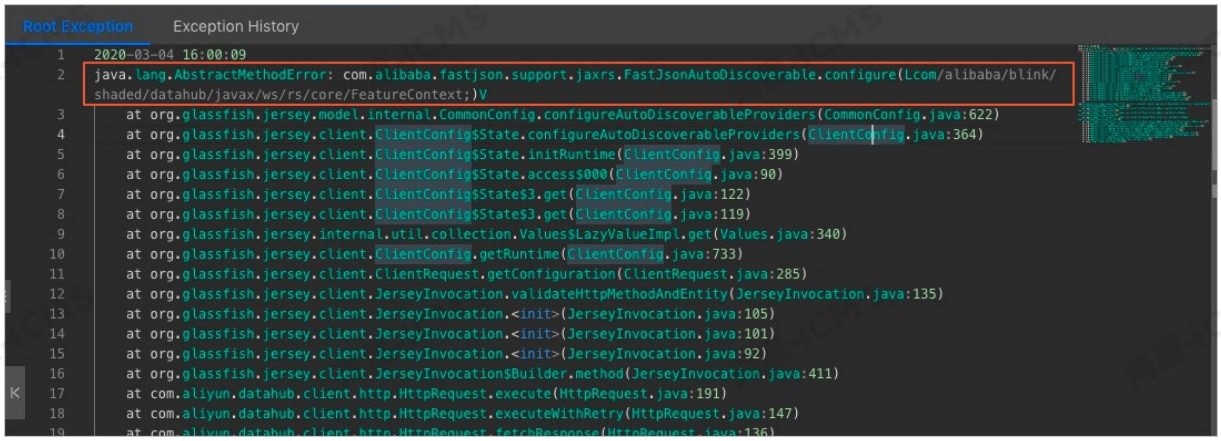
   If information similar to that in the following figure appears, Realtime Compute for Apache Flink writes the DataHub data to ApsaraDB for HBase.

```
hbase(main):128:0> scan 'hbase_sink'
ROW                    COLUMN+CELL
 rowkey3               column=f1:a, timestamp=1597741134871, value=[B@56bc3604
 rowkey3               column=f1:b, timestamp=1597741134871, value=[B@f5c05
 rowkey3               column=f1:c, timestamp=1597741134871, value=[B@25c03326
1 row(s)
Took 0.0590 seconds
```

FAQ

If an error similar to the following one appears when a job is running, a JAR package conflict occurs. What do I do?

```
java.lang.AbstractMethodError:com.alibaba.fastjson.support.jaxrs.FastJsonAutoDiscoverable
onfigure(Lcom/alibaba/blink/shaded/datahub/javax/ws/rs/core/FeatureContext;)
```



We recommend that you use the relocation feature of **maven-shade-plugin** to resolve the JAR package conflict.

```
<relocations combine.self="override">
   <relocation>
       <pattern>org.glassfish.jersey</pattern>

<shadedPattern>com.alibaba.blink.shaded.datahub.org.glassfish.jersey</shadedPattern>
   </relocation>
</relocations>
```

7.8.4. Read data from Log Service

This topic describes how to run a Flink DataStream job to read data from Log Service.

Prerequisites

- Java Development Kit (JDK) 8 is installed on your on-premises machine.
- Maven 3.x is installed on your on-premises machine.

- An integrated development environment (IDE) for Java or Scala is installed on your on-premises machine. We recommend that you use IntelliJ IDEA. The JDK and Maven are configured.
- A Logstore is created in Log Service, and test data exists in the Logstore.

Background information

Windows OS is used in this demo.

> **Important** Only Blink 3.X supports this demo.

Develop a job

1. Download and decompress the SLS_Demo-master.zip package to your on-premises machine.
2. In IntelliJ IDEA, choose **File > Open** to open the decompressed SLS_Demo-master folder.
3. Double-click the **ConsumerSample.java** file in the \SLS_Demo-master\src\main\java\com\aliyun\openservices\log\flink directory. Then, configure the parameters related to Log Service in the ConsumerSample.java file.

```
    private static final String SLS_ENDPOINT = "VPC endpoint";// Use the classic network endpoint or a virtual private cloud (VPC) endpoint in the production environment.
//  private static final String SLS_ENDPOINT = "public endpoint";// Use the public endpoint in the test environment.
    private static final String ACCESS_KEY_ID = "yourAK";
    private static final String ACCESS_KEY_SECRET = "yourAS";
    private static final String SLS_PROJECT = "yourProject";
    private static final String SLS_LOGSTORE = "yourlogstore";
    // 1. Specify the start offset, which indicates the timestamp to start reading data from Log Service. The timestamp is measured in seconds. 2. To read both full and incremental data from Log Service, set the StartInMs parameter to Consts.LOG_BEGIN_CURSOR.
    // 3. To read only incremental data from Log Service, set the StartInMs parameter to Consts.LOG_END_CURSOR.
    private static final String StartInMs = Consts.LOG_END_CURSOR;
```

> **Note** You must comment out <scope>provided</scope> when you perform local debugging in your IDE.

4. Go to the directory where the **pom.xml** file is stored. Then, run the following command to package the file:

```
mvn clean package
```

A Java Archive (JAR) package named **flink-log-connector-0.1.21-SNAPSHOT.jar** appears in the target directory, based on the artifactId parameter that you configured in the pom.xml file for your project. This indicates that job development is complete.

Publish a job

For more information about how to publish a job, see Publish a job.

> **Note** Before you publish the job, set the Parallelism parameter for the source table on the **Configurations** tab of the **Development** page. The parallelism setting of the source table cannot be greater than the number of shards in the source table. Otherwise, a JobManager error occurs after the job starts.

The following example shows the job content:

```
-- Required. The full name of the main class.
--blink.main.class=com.aliyun.openservices.log.flink.ConsumerSample

-- The name of the job.
blink.job.name=sls

-- The resource name of the JAR package that contains the full name of the main class.
If multiple JAR packages exist, you must specify this parameter.
--blink.main.jar=flink-log-connector-0.1.21-snapshot.jar

-- The default state backend configuration. This field takes effect when the job code i
s not explicitly configured.
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000

-- The default checkpoint configuration. The configuration takes effect when the job co
de is not explicitly configured.
blink.checkpoint.interval.ms=180000
```

> **Note** You can configure custom parameters. For more information, see Set custom parameters.

Verify the test results

On the Container Log tab of the **Job Administration** page, view information in the **taskmanager.out** file of the sink node. In this example, the type of the sink node is print.

If information shown in the following figure appears, Realtime Compute for Apache Flink reads the data from Log Service.

8.Best Practices

8.1. Best practices of Realtime Compute in the e-commerce industry

8.1.1. Real-time PV and UV curves in e-commerce scenarios

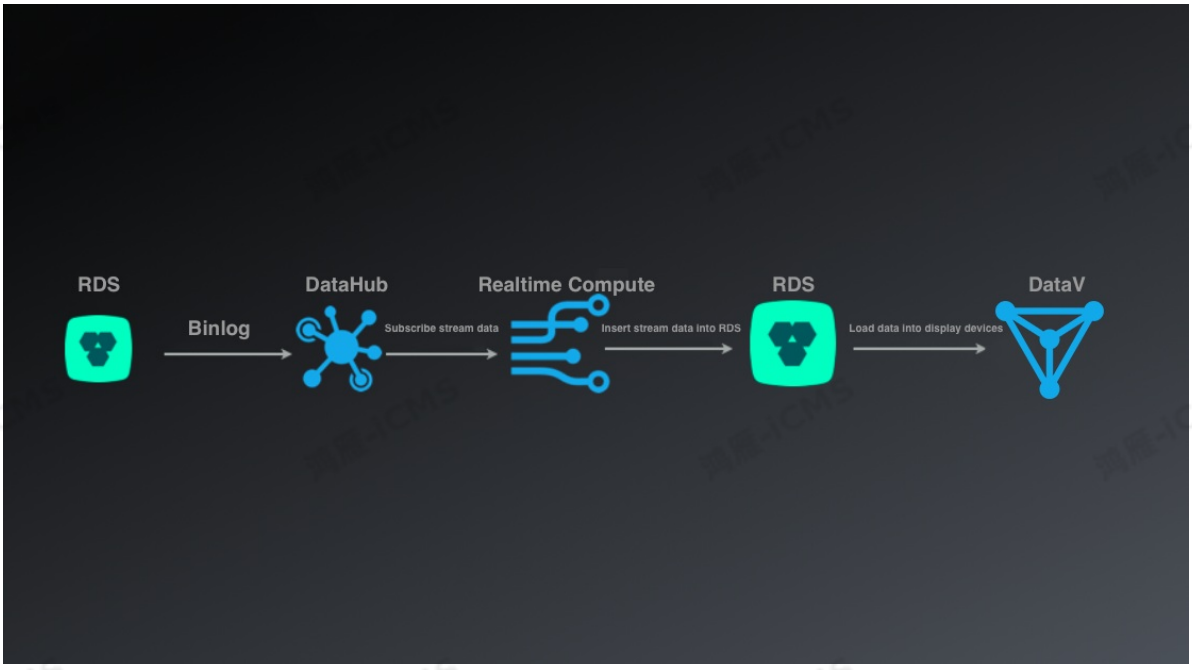
This topic provides a use case of Gegejia, a partner of Realtime Compute for Apache Flink, to describe how to use Realtime Compute for Apache Flink to create real-time page view (PV) and unique visitor (UV) curves.

Background information

As the new retail industry rises, competition in the Internet e-commerce industry is becoming increasingly fierce. Real-time data is particularly important to the e-commerce industry, such as collecting statistics on the total PVs and UVs to a website.

Example

- Business architecture



- Workflow
  i. The SDK provided by DataHub synchronizes binary logs to DataHub.
  ii. Realtime Compute for Apache Flink subscribes to data in DataHub for real-time computing.
  iii. Realtime Compute for Apache Flink writes real-time data to ApsaraDB RDS.
  iv. Alibaba Cloud DataV or other data visualization service presents the result data.

- Preparations Table 1. Fields in a log source table

Field	Data type	Description
account_id	VARCHAR	The user ID.
client_ip	VARCHAR	The IP address of the client.
client_info	VACHAR	The model of the device.
platform	VARHCAR	The operating system type of the device.
imei	VARCHAR	The International Mobile Equipment Identity (IMEI) number of the device.
version	BIGINT	The operating system version of the device.
action	BIGINT	The page redirection description.
gpm	VARCHAR	The tracking path.
c_time	VARCHAR	The time at which the request was made.
target_type	VARCHAR	The type of requested data.
target_id	VARCHAR	The ID of requested data.
udata	VARCHAR	The extended information.
session_id	VARHCAR	The session ID
product_id_chain	VARHCAR	The string of product IDs.
cart_product_id_chain	VARCHAR	The ID string of the products added to the cart.
tag	VARCHAR	The special tag.
position	VARCHAR	The location of the user.
network	VARCHAR	The network type of the user.
p_dt	VARCHAR	The time-based partition, in days.
p_platform	VARCHAR	The partition system version.

Table 2. Fields in an ApsaraDB RDS result table

Field	Data type	Description
summary_date	BIGINT	The date on which the statistics are collected.

summary_min	VARCHAR	The minute at which the statistics are collected.
pv	BIGINT	The number of clicks on the specified website.
uv	BIGINT	The number of visitors who click the specified website. **Note** Only one UV is counted for multiple clicks by the same visitor within one day.
currenttime	TIMESTAMP	The current system time.

- Business logic

```
// Create an order source table.
CREATE TABLE source_ods_fact_log_track_action (
    account_id                      VARCHAR, // The ID of the user.
    client_ip                       VARCHAR, // The IP address of the client.
    client_info                     VARCHAR, // The model of the device.
    platform                        VARCHAR, // The operating system type of the dev
ice.
    imei                            VARCHAR, // The IMEI number of the device.
    `version`                       VARCHAR, // The operating system version of the
device.
    `action`                        VARCHAR, // The page redirection description.
    gpm                             VARCHAR, // The tracking path.
    c_time                          VARCHAR, // The time at which the request was ma
de.
    target_type                     VARCHAR, // The type of the requested data.
    target_id                       VARCHAR, // The ID of the requested data.
    udata                           VARCHAR, // The extended information in the JSON
format.
    session_id                      VARCHAR, // The ID of the session.
    product_id_chain                VARCHAR, // The ID string of products.
    cart_product_id_chain           VARCHAR, // The ID string of the products added
to the cart.
    tag                             VARCHAR, // The special tag.
    `position`                      VARCHAR, // The location of the user.
    network                         VARCHAR, // The network type of the user.
    p_dt                            VARCHAR, // The time-based partition, in days.
    p_platform                      VARCHAR // The partition system version.
) WITH (
    type='datahub',
    endPoint='yourEndpointURL',
    project='yourProjectName',
    topic='yourTopicName',
    accessId='yourAccessId',
    accessKey='yourAccessSecret',
    batchReadSize='1000'
);
```

```
;

CREATE TABLE result_cps_total_summary_pvuv_min (
    summary_date              bigint, // The date on which the statistics are collected.
    summary_min               varchar, // The minute at which the statistics are collected.
    pv                        bigint, // The number of clicks on the specified website.
    uv                        bigint, // The number of visitors who click the specified website. Only one UV is counted for multiple clicks by the same visitor within one day.
    currenttime               timestamp, // The current system time.
    primary key (summary_date,summary_min)
) WITH (
    type= 'rds',
    url = 'yourRDSDatabaseURL',
    userName = 'yourDatabaseUserName',
    password = 'yourDatabasePassword',
    tableName = 'yourTableName'
);

CREATE VIEW result_cps_total_summary_pvuv_min_01 AS
select
cast(p_dt as bigint) as summary_date // The time-based partition, in days.
,count(client_ip) as pv // Count the number of PVs by the client IP address.
,count(distinct client_ip) as uv // Count the number of UVs by deduplicating client IP addresses.
,cast(max(c_time ) as TIMESTAMP)  as c_time // The time at which the request was made.
from source_ods_fact_log_track_action
group by p_dt;

INSERT into  result_cps_total_summary_pvuv_min
select
a.summary_date, // The time-based partition, in days.
cast(DATE_FORMAT(c_time,'HH:mm')  as varchar) as summary_min, // Obtain the time string representing the hour and minute.
a.pv,
a.uv,
CURRENT_TIMESTAMP  as currenttime // The current system time.
from result_cps_total_summary_pvuv_min_01 AS a
;
```

- Key points

  To help you understand structured code and facilitate code maintenance, we recommend that you use views to split the business logic into two modules. For more information about views, see Create a data view.

- Module 1

```
CREATE VIEW result_cps_total_summary_pvuv_min_01 AS
select
cast(p_dt as bigint) as summary_date // The time-based partition, in days.
,count(client_ip) as pv // Count the number of PVs by client IP address.
,count(distinct client_ip) as uv // Count the number of UVs by deduplicating visitors.
,cast(max(c_time ) as TIMESTAMP)  as c_time // The time at which the request was made.
from source_ods_fact_log_track_action
group by p_dt;
```

- PV is the number of clicks after a customer visits the website, and UV is the number of unique visitors after customer IP addresses are deduplicated.
- **cast(max(c_time) as TIMESTAMP)** specifies the time at which the last request was made.
- **p_dt** is used as the time-based partition, and the unit is day. **max(c_time)** is used as the deadline for visiting a website, and a PV and UV are inserted into the database.

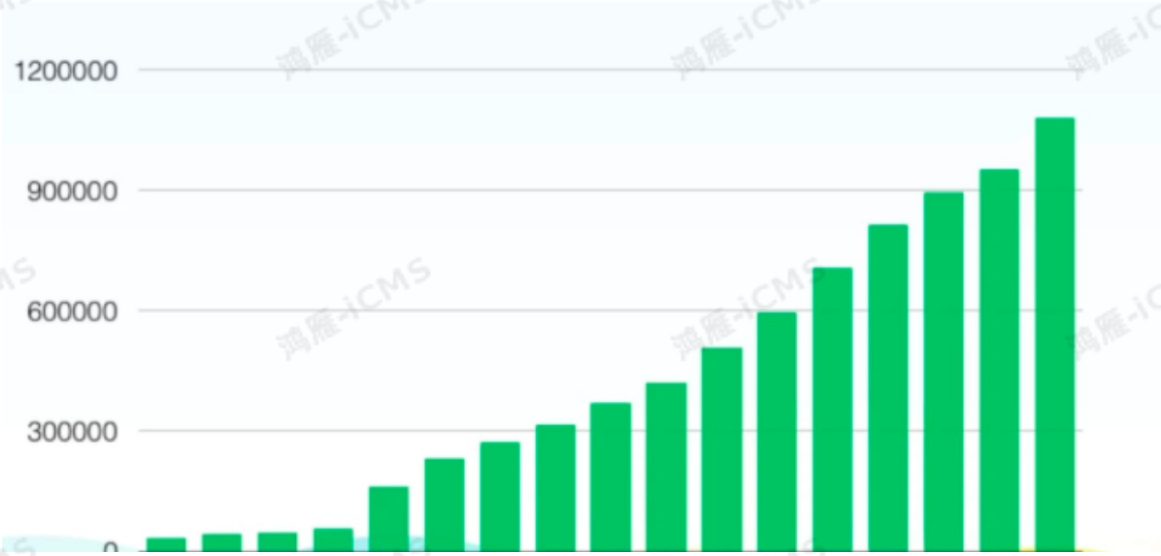
Table 3. Result

p_dt	pv	uv	max(c_time)
2017-12-12	1000	100	2017-12-12 9:00:00
2017-12-12	1500	120	2017-12-12 9:01:00
2017-12-12	2200	200	2017-12-12 9:02:00
2017-12-12	3300	320	2017-12-12 9:03:00

- Module 2

```
INSERT into  result_cps_total_summary_pvuv_min
select
a.summary_date, // The time-based partition, in days.
cast(DATE_FORMAT(c_time,'HH:mm')  as varchar) as summary_min, // Obtain the time st
ring that represents the hour and minute.
a.pv,
a.uv,
CURRENT_TIMESTAMP  as currenttime // The current system time.
from result_cps_total_summary_pvuv_min_01 AS a;
```

Extract the data from module 1 by hour and minute and obtain the PV and UV growth curves,



Example and source code

Based on the PV and UV curve solution described in this topic, Alibaba Cloud creates a demo that includes a complete link for you. You can use this demo to register upstream and downstream storage resources and obtain your PV and UV curves. You can click sample code to download the complete demo. Take note of the following two points when you use this demo to register upstream and downstream storage resources:

- Use a DataHub table as the source table.
- Create an ApsaraDB RDS result table.

8.1.2. Marketing coupons in e-commerce scenarios

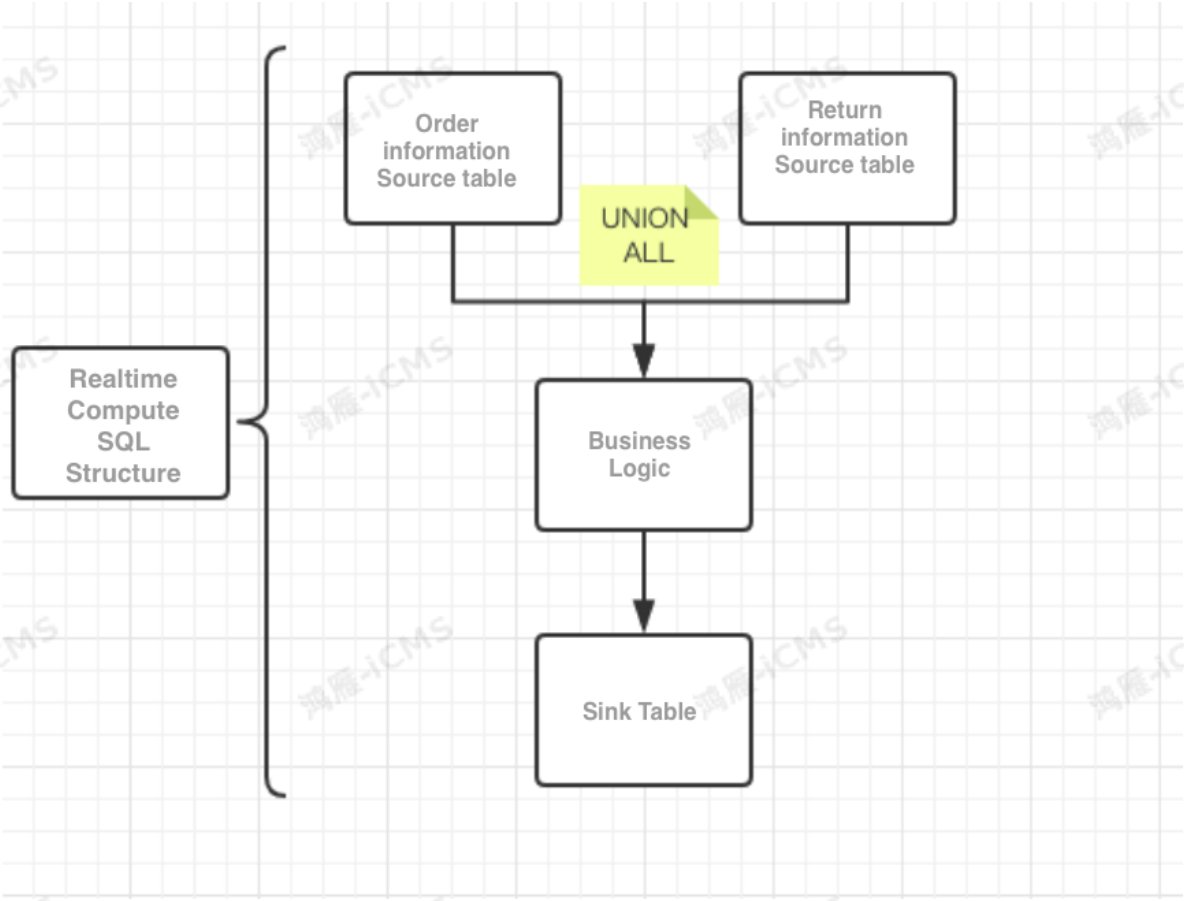
This topic describes how to use Realtime Compute for Apache Flink to filter users who meet the conditions for issuing marketing coupons in a coupon-based marketing policy.

Background information

A merchant uses a marketing policy of refund coupons at Double 11. After the consumption amount of a user reaches a specified value, the merchant issues a refund coupon with a specific amount to the user to promote more consumption. Realtime Compute for Apache Flink monitors the consumption amount of users in real time and filters users who meet the conditions for issuing refund coupons.

Solution

- SQL structure



- Source tables

- Create a source table to store orders generated by the e-commerce system.

  > **Note** The order data format is simplified to focus on the core logic. Only the attributes related to the use case are retained.

```
CREATE TABLE dwd_tb_trd_pay_ri(
        biz_order_id    VARCHAR, -- 'Order ID'
        auction_id      VARCHAR, -- 'Product ID'
        auction_title   VARCHAR, -- 'Product title'
        buyer_id        VARCHAR, -- 'ID of the buyer'
        buyer_nick      VARCHAR, -- 'Nickname of the buyer'
        pay_time        VARCHAR, -- 'Payment time of the order'
        gmt_create      VARCHAR, -- 'Time at which the order was created'
        gmt_modified    VARCHAR, -- 'Time at which the order was modified'
        biz_type        VARCHAR, -- 'Transaction type'
        pay_status      VARCHAR, -- 'Payment status'
        `attributes`    VARCHAR, -- 'Flag of the order'
        from_group      VARCHAR, -- 'Source of the order'
        div_idx_actual_total_fee   DOUBLE  --'Transaction amount'
) WITH (
    type='datahub',
    endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
    project='yourProjectName',-- 'Name of your project'
    topic='yourTopicName',--'Name of your topic'
    roleArn='yourRoleArn',-- 'Alibaba Cloud Resource Name (ARN) of your role'
    batchReadSize='500'
);
```

- Create a source table of returns.

  > **Note** The order data format is simplified to focus on the core logic. Only the attributes related to the use case are retained.

```
CREATE TABLE dwd_tb_trd_rfd_ri(
        biz_order_id    VARCHAR, -- 'Order ID'
        auction_id      VARCHAR, -- 'Product ID'
        auction_title   VARCHAR, -- 'Product title'
        buyer_id        VARCHAR, -- 'ID of the buyer'
        buyer_nick      VARCHAR, -- 'Nickname of the buyer'
        pay_time        VARCHAR, -- 'Payment time of the order'
        gmt_create      VARCHAR, -- 'Time at which the order was created'
        gmt_modified    VARCHAR, -- 'Time at which the order was modified'
        biz_type        VARCHAR, -- 'Transaction type'
        pay_status      VARCHAR, -- 'Payment status'
        `attributes`    VARCHAR, -- 'Flag of the order'
        from_group      VARCHAR, -- 'Source of the order'
        refund_fee      DOUBLE   -- 'Refund amount'
) WITH (
    type='datahub',
    endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
    project='yourProjectName', --'Your project'
    topic='yourTopicName', --'Your topic'
    roleArn='yourRoleArn', --'ARN of your role'
    batchReadSize='500'
);
```

- Result tables

  Execute the following statements to create an ApsaraDB RDS result table:

```
CREATE TABLE tddl_output(
    gmt_create   VARCHAR, --'Time at which the order was created'
    gmt_modified VARCHAR, --'Time at which the order was modified'
    buyer_id     BIGINT, --'ID of the buyer'
    cumulate_amount BIGINT, --'Transaction amount'
    effect_time BIGINT, --'Payment time of the order'
    primary key(buyer_id,effect_time)
) WITH (
        type= 'rds',
        url = 'yourDatabaseURL',  --'URL of your database'
        tableName = 'yourTableName', --'Name of your table'
        userName = 'yourUserName',  --'Your username'
        password = 'yourDatabasePassword' --'Your password'
    );
```

- Business logic

  Perform the UNION ALL operation to join the order table and table of returns to obtain information about all purchased items and collect the actual consumption amount and details of users.

```
CREATE VIEW  dwd_tb_trd_and_rfd_pay_ri
AS
SELECT
*
FROM(
    (SELECT
        `a`.biz_order_id biz_order_id,
        `a`.auction_id auction_id,
        `a`.auction_title auction_title,
        `a`.buyer_id buyer_id,
        `a`.buyer_nick buyer_nick,
        `a`.pay_time pay_time,
        `a`.gmt_create gmt_create,
        `a`.gmt_modified gmt_modified,
        `a`.biz_type biz_type,
        `a`.pay_status pay_status,
        `a`.`attributes` `attributes`,
        `a`.from_group,
        `a`.div_idx_actual_total_fee div_idx_actual_total_fee
    FROM
        dwd_tb_trd_pay_ri `a`
    )
    UNION ALL
    (
    SELECT
        `b`.biz_order_id biz_order_id,
        `b`.auction_id auction_id,
        `b`.auction_title auction_title,
        `b`.buyer_id buyer_id,
        `b`.buyer_nick buyer_nick,
        `b`.pay_time pay_time,
        `b`.gmt_create gmt_create,
        `b`.gmt_modified gmt_modified,
        `b`.biz_type biz_type,
        `b`.pay_status pay_status,
        `b`.`attributes` `attributes`,
        `b`.from_group,
        `b`.refund_fee div_idx_actual_total_fee --The refund amount, which is a
negative value.
    FROM
        dwd_tb_trd_rfd_ri `b`
    )
);
```

- Deduplication

  A large amount of duplicate data such as product IDs and product names may exist in the order table and return table. Use the MIN function to obtain parameter values only in the first generated record of each order and filter out other data. Then, group data by order ID and payment status to obtain required product IDs and product names.

```
CREATE VIEW filter_hxhb_dwd_tb_trd_and_rfd_pay_ri_distinct AS
SELECT
   biz_order_id biz_order_id,
   pay_status pay_status,
   MIN(auction_id) auction_id,
   MIN(auction_title) auction_title,
   MIN(buyer_id) buyer_id,
   MIN(buyer_nick)  buyer_nick,
   MIN(pay_time)  pay_time,
   MIN(gmt_create)  gmt_create,
   MIN(gmt_modified)  gmt_modified,
   MIN(biz_type)  biz_type,
   MIN(attributes)  attributes,
   MIN(div_idx_actual_total_fee)  div_idx_actual_total_fee
FROM
    dwd_tb_trd_and_rfd_pay_ri
GROUP BY biz_order_id ,pay_status;
```

- Data aggregation

```
CREATE VIEW ads_tb_trd_and_rfd_pay_ri AS
SELECT
   MIN(gmt_create) gmt_create,  --'Time at which the order was created'
   MAX(gmt_modified) gmt_modified,  --'Time at which the order was last modified'
   CAST (buyer_id AS BIGINT) buyer_id,  --'ID of the buyer'
   CAST (date_format(pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMdd') AS BIGINT) as eff
ect_time, --'Payment time'
   SUM(CAST(div_idx_actual_total_fee/100 AS BIGINT)) cumulate_amount  --'Transaction
amount'
FROM
   filter_hxhb_dwd_tb_trd_and_rfd_pay_ri_distinct
GROUP BY
   buyer_id,date_format(pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMdd');
```

Q: Why are the MAX and MIN functions used?

```
 MIN(gmt_create) gmt_create, --'Time at which the order was created'
 MAX(gmt_modified) gmt_modified, --'Time at which the order was last modified'
```

A: **MIN(gmt_create)** obtains the time at which an order was created. **MAX(gmt_modified)** obtains the time at which the order was last modified. You can use MAX and MIN to obtain the time values based on the business logic of orders.

> **Note** If the time field is not of the BIGINT type, use the related built-in functions to convert the data type. For more information, see Built-in functions.

- Data insertion into ApsaraDB RDS

Insert the statistical data into the ApsaraDB RDS result table. Then, use appropriate push software to issue coupons with the correct refund amount to users who meet the specified conditions.

```
INSERT INTO tddl_output
SELECT
    gmt_create,
    gmt_modified,
    buyer_id,
    cumulate_amount,
    effect_time
from ads_tb_trd_and_rfd_pay_ri
where cumulate_amount>0;
```

Q: Why is the "cumulate_amount>0" clause used?

```
cumulate_amount>0
```

A: This clause filters out the amount of the returned products involved in the preceding UNION ALL operation.

Summary

Q: How do I obtain the data I need from a large number of records about orders and return orders?

A: In actual transaction situations, a large number of order records and return records are generated. You can perform the UNION ALL operation to join two or more tables as one large table. Then, deduplicate and aggregate the records based on your specific business logic. Finally, write the actual transaction amounts of all orders of each user into the result table to prepare for issuing coupons.

Demo example and source code

Based on the marketing refund solution described in this topic, Alibaba Cloud creates a demo that includes a complete process for you.

- Use a DataHub table as the source table.
- Create an ApsaraDB RDS result table.

You can refer to this demo to register upstream and downstream storage resources and develop your marketing refund solutions. Click the demo code to download it.

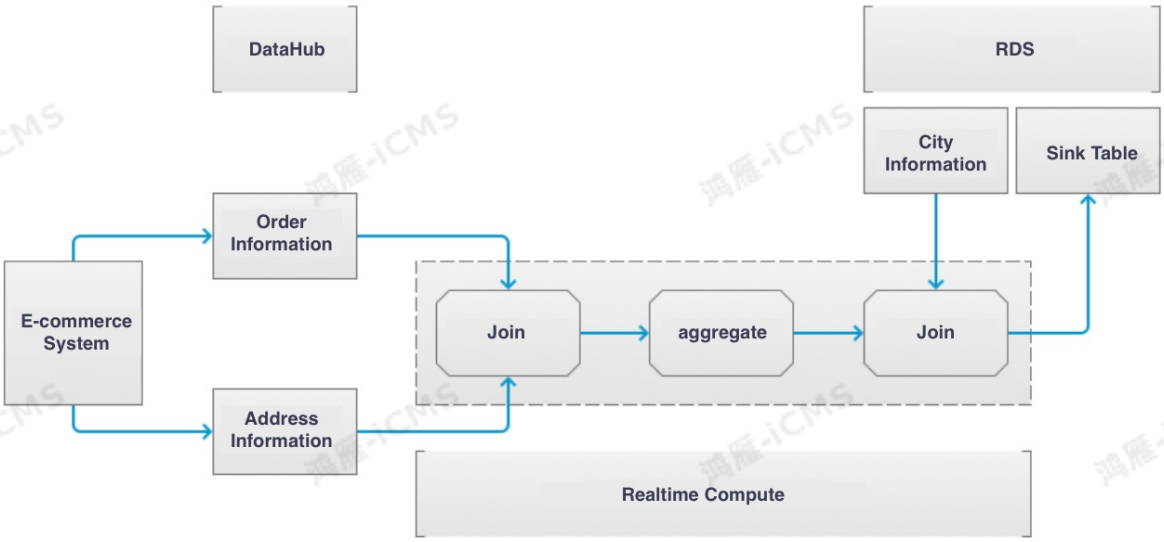
8.1.3. Real-time situation awareness and geographic distribution of orders in e-commerce scenarios

This topic provides a use case to describe how to use Realtime Compute to implement real-time situation awareness and geographic distribution of orders.

Background

Real-time situation awareness and geographic order distribution help enterprises optimize the allocation and release of product categories in a timely manner. The following use case describes how a food e-commerce enterprise uses Realtime Compute to implement real-time situation awareness and geographic distribution of orders.

Use case



Note To focus on the core logic, we simplify the order data format to retain only the attributes related to the use case.

- Create multiple tables to store data.

  In the e-commerce system, orders and shipping addresses are separately stored. A buyer can use multiple shipping addresses to place orders. No shipping address is specified when an order is created. The specific shipping address can be obtained only when the order is submitted. The shipping address table stores city IDs (specified by city_id). You also need to create a city table to store the geographic information about cities. We create these tables to collect statistics on the distribution of orders (GMV) in different regions by day.

  - Create an order table.

```
CREATE TABLE source_order (
    id VARCHAR, // The ID of the order.
    seller_id VARCHAR, // The ID of the seller.
    account_id VARCHAR, // The ID of the buyer.
    receive_address_id VARCHAR, // The ID of the shipping address.
    total_price VARCHAR, // The order amount.
    pay_time VARCHAR, // The payment time of the order.
) WITH (
    type='datahub',
    endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
    project='yourProjectName', // Your project name.
    topic='yourTopicName', // Your topic name.
    roleArn='yourRoleArn', // Your role ARN.
    batchReadSize='500'
);
```

- Create a shipping address table.

```
CREATE TABLE source_order_receive_address (
     id VARCHAR, // The ID of the shipping address.
     full_name VARCHAR, // The full name of the consignee.
     mobile_number VARCHAR, // The mobile number of the consignee.
     detail_address VARCHAR, // The detailed shipping address.
     province VARCHAR, // The province in the shipping address.
     city_id VARCHAR, // The city in the shipping address.
     create_time VARCHAR // The time when the order was created.
 ) WITH (
     type='datahub',
     endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
     project='yourProjectName', // Your project name.
     topic='yourProjectName', // Your topic name.
     roleArn='yourProjectName', // Your role ARN.
     batchReadSize='500'
 );
```

- Create a city table.

```
CREATE TABLE dim_city (
     city_id varchar,
     city_name varchar, // The name of the city.
     province_id varchar, // The ID of the province to which the city belongs.
     zip_code varchar, // The postal code.
     lng varchar, // The longitude of the city.
     lat varchar, // The latitude of the city.
  PRIMARY KEY (city_id),
  PERIOD FOR SYSTEM_TIME // Specify that this is a dimension table.
 ) WITH (
     type= 'rds',
     url = 'yourDatabaseURL', // Your database URL.
     tableName = 'yourTableName', // Your table name.
     userName = 'yourDatabaseName', // Your username.
     password = 'yourDatabasePassword' // Your password.
 );
```

- Collect statistics on the distribution of orders (GMV) in different regions by day.

```
CREATE TABLE result_order_city_distribution (
     summary_date bigint, // The date when the statistics are collected.
     city_id bigint, // The ID of the city.
     city_name varchar, // The name of the city.
     province_id bigint, // The ID of the province to which the city belongs.
     gmv double, // The GMV.
     lng varchar, // The longitude of the city.
     lat varchar, // The latitude of the city.
     primary key (summary_date,city_id)
    ) WITH (
        type= 'rds',
        url = 'yourDatabaseURL', // Your database URL.
        tableName = 'yourTableName', // Your table name.
        userName = 'yourDatabaseName', // Your username.
        password = 'yourDatabasePassword' // Your password.
    );
```

- Edit business logic.

```
 insert into result_order_city_distribution
 select
 d.summary_date
 ,cast(d.city_id as BIGINT)
 ,e.city_name
 ,cast(e.province_id as BIGINT)
 ,d.gmv
 ,e.lng
 ,e.lat
 ,e.lnglat
 from
 (
         select
         DISTINCT
         DATE_FORMAT(a.pay_time,'yyyyMMdd') as summary_date
         ,b.city_id as city_id
         ,round(sum(cast(a.total_price as double)),2) as gmv
         from source_order as a
         join source_order_receive_address as b on a.receive_address_id =b.id
         group by DATE_FORMAT(a.pay_time,'yyyyMMdd'),b.city_id
         // Join the order table with the shipping address table and compute the sales distribution by date and city ID.
 )d join dim_city FOR SYSTEM_TIME AS OF PROCTIME() as e on d.city_id = e.city_id
 // Join the dimension table to complete the geographic information about cities and obtain the final computing result.
 ;
```

8.1.4. Latest transaction records in e-commerce scenarios

This topic describes how to use Realtime Compute to obtain the latest transaction records.

Background

The order transaction table of an e-commerce merchant may contain operation records about the same order at different times. For example, the order transaction table records a user's operations such as modifying the quantity of the product to be ordered and returning the product. The merchant only wants to obtain valid transaction records. The following section describes how to use Realtime Compute to obtain the latest transaction records.

Procedure

- 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]
```

Parameter	Description
`ROW_NUMBER()`	The OVER window function used to compute the number of a row. The value starts from 1.
`PARTITION BY col1[, col2..]`	The columns by which you want to partition the table. This parameter is optional.
`ORDER BY col1 [asc|desc][, col2 [asc|desc]...]`	The columns by which you want to sort your data. You can specify different orders for multiple columns.

- Examples
  - Test data

id (BIGINT)	TIME (VARCHAR)	VALUE (BIGINT)
1	1517898096	5
1	1517887296	44
1	1517872896	32
2	1517872896	10

- Test statements

```
create table source_table (
  id      BIGINT,
  `TIME`  VARCHAR,
  `VALUE` BIGINT
)with(
  type='datahub',
  endPoint='yourEndpointURL',
  project='yourProjectName',
  topic='yourTableName',
  accessId='yourAccessId',
  accessKey='yourAccessSecret'
);
CREATE TABLE result_table (
     id   BIGINT,
  `TIME`  VARCHAR,
  `VALUE` BIGINT
) WITH (
    type= 'rds',
    url = 'yourRDSDatabaseURL',
    userName = 'yourDatabaseName',
    password = 'yourDatabasePassword',
    tableName = 'yourTableName'
);
INSERT INTO result_table
SELECT id,`TIME`,`VALUE`
FROM (
    SELECT *,
       ROW_NUMBER() OVER (PARTITION BY id ORDER BY `TIME` desc) AS rownume
    FROM
    source_table
)
WHERE rownume <= 1
;
```

- Test results

id (BIGINT)	TIME (VARCHAR)	VALUE (BIGINT)
1	1517898096	5
2	1517872896	10

- Key points

```
SELECT id,`TIME`,`VALUE`
FROM (
   SELECT *,
      ROW_NUMBER() OVER (PARTITION BY id ORDER BY `TIME` desc) AS rownume
   FROM
   source_table
)
WHERE rownume <= 1
```

Some orders in the order table have multiple records generated at different times. To obtain only the last generated record for each order, use `row_number() OVER (PARTITION BY id ORDER BY TIME DESC)` to group records by order ID and sort the records in each group in descending order of time. The computing result lists the sequential numbers (which are continuous and unique in a group) of sorted records in each group. The first record in each group is the last generated record of the corresponding order.

8.2. Best practices of Realtime Compute in the live streaming industry

8.2.1. Core video metric monitoring for live streaming

This topic provides a use case to describe how to use Realtime Compute for Apache Flink to monitor core video metrics.

Background information

As the Internet technologies evolve, live streaming, especially the live streaming ecological chain, attracts more and more attention. Live streaming allows viewers to watch various videos, such as sports events, major events, and news, online and in real time over the Internet.

Poor user experience leads to a loss of users. Therefore, live streaming platforms must focus on the following points:

- Quality of experience (QoE) of casters and audiences: Focus on system metrics, such as frame freezing rate of video or audio signals, delay rate, and packet loss rate.
- Timeliness: Identify issues related to the system in real time, and locate the issues in advance.
- Overall customer operation of the website: track user trends and identify popular videos

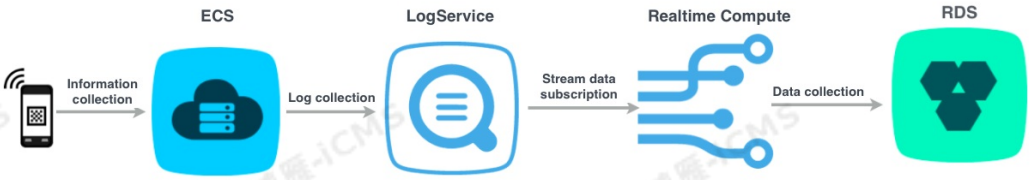
This topic provides a use case to describe how to use Realtime Compute for Apache Flink to monitor the system stability and the operations of a live streaming platform.

Description

To build a highly interactive social community and cover more live streaming scenarios to realize more profit, a platform operator can take the following actions:

- Hire multiple casters for its live streaming website.
- Allow each caster to stream to audiences in a channel.
- Allow users to watch the video of the caster in the current channel and hear the voice of the caster.
- Allow each caster to invite multiple audiences in a channel to a private chat.

Figure 1. Workflow



1. Casters and audiences use a live streaming app that sends streaming data to the server every 10 seconds.
2. After the server receives data from the app, the server saves the data to its local disk. Alibaba Cloud Log Service then collects the data from the server.
3. Realtime Compute for Apache Flink subscribes to the data on Log Service.
4. Realtime Compute for Apache Flink analyzes live streaming data.

Business objectives

Obtain the following metrics based on the tracked logs sent from the client app:

- Channel faults, including frame freezing, frame drop, and out of synchronization between the audio and video signals
- Average end-to-end latency collected by region
- Overall frame freezing rate collected in real time (Number of online users who encounter frame freezing/Total number of online users × 100%. This metric can be used to measure the scope of users who encounter frame freezing.)
- Average number of frame freezing times per user (Total number of times online frame freezing occurs/Total number of online users. This metric can be used to measure the overall severity of frame freezing based on frame freezing times.)

It is expected that the preceding data is calculated and written to an ApsaraDB RDS database in real time. This way, online data and even some alerts can be displayed in reports and dashboards.

Data format

The following table describes the data format of the tracked logs that the app client sends to the server.

Field name	Description
ip	The IP address of the client.
agent	The type of the client.
roomid	The ID of the channel.
userid	The user ID.
abytes	The audio bitrate.
afcnt	The number of audio frames.
adrop	The number of dropped audio frames.
afts	The audio timestamp.

alat	The end-to-end latency of audio frames.
vbytes	The video bitrate.
vfcnt	The number of video frames.
vdrop	The number of dropped video frames.
vfts	The video timestamp.
vlat	The end-to-end latency of video frames.
ublock	The number of upstream frame freezing times.
dblock	The number of downstream frame freezing times.
timestamp	The timestamp when a log was generated.
region	The region where live streaming is performed.

Log Service uses semi-structured storage and displays the preceding fields in the following log format:

```
{
    "ip": "ip",
    "agent": "agent",
    "roomid": "123456789",
    "userid": "123456789",
    "abytes": "123456",
    "afcnt": "34",
    "adrop": "3",
    "afts": "1515922566",
    "alat": "123",
    "vbytes": "123456",
    "vfcnt": "34",
    "vdrop": "4",
    "vfts": "1515922566",
    "vlat": "123",
    "ublock": "1",
    "dblock": "2",
    "timestamp": "15151922566",
    "region": "hangzhou"
}
```

SQL statements

- Data cleansing
  Declare the source table in Realtime Compute for Apache Flink.

```
CREATE TABLE app_heartbeat_stream_source (
    ip VARCHAR,
    agent VARCHAR,
    roomid VARCHAR,
    userid VARCHAR,
    abytes VARCHAR,
    afcnt VARCHAR,
    adrop VARCHAR,
    afts VARCHAR,
    alat VARCHAR,
    vbytes VARCHAR,
    vfcnt VARCHAR,
    vdrop VARCHAR,
    vfts VARCHAR,
    vlat VARCHAR,
    ublock VARCHAR,
    dblock VARCHAR,
    `timestamp` VARCHAR,
    app_ts AS TO_TIMESTAMP(CAST(`timestamp` AS BIGINT)), // Specify the fields for wh
ich a watermark is generated.
    WATERMARK FOR app_ts AS withOffset(app_ts, 10000) // Add an offset of 10 seconds
to define a watermark.
) WITH (
    type ='sls',
    endPoint ='http://cn-hangzhou-corp.sls.aliyuncs.com',
    accessId ='xxxxxxxxxx',
    accessKey ='xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    project ='xxxx',
    logStore ='app_heartbeat_stream_source',
);
```

For business convenience, all data is processed as the VARCHAR type in the source table. For subsequent processing, data in the source table is cleaned for the following purposes:

i. Format conversion: Convert some data of the VARCHAR type to the BIGINT type.

ii. Business data supplement: For example, enter region-related information.

```
CREATE VIEW view_app_heartbeat_stream AS
SELECT
    ip,
    agent,
    CAST(roomid AS BIGINT),
    CAST(userid AS BIGINT),
    CAST(abytes AS BIGINT),
    CAST(afcnt AS BIGINT),
    CAST(adrop AS BIGINT),
    CAST(afts AS BIGINT),
    CAST(alat AS BIGINT),
    CAST(vbytes AS BIGINT),
    CAST(vfcnt AS BIGINT),
    CAST(vdrop AS BIGINT),
    CAST(vfts AS BIGINT),
    CAST(vlat AS BIGINT),
    CAST(ublock AS BIGINT),
    CAST(dblock AS BIGINT),
    app_ts,
    region
FROM
    app_heartbeat_stream_source;
```

- Channel fault statistics
  Use a new window every 10 minutes to collect statistics on channel faults, including frame freezing, frame drop, and out of synchronization between the audio and video signals.

```
CREATE VIEW room_error_statistics_10min AS
SELECT
    CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
    roomid,
    SUM(ublock) as ublock, // Collect statistics on the number of upstream frame freezing times in the last 10 minutes.
    SUM(dblock) as dblock, // Collect statistics on the number of downstream frame freezing times in the last 10 minutes.
    SUM(adrop) as adrop, // Collect statistics on the number of audio packets dropped in the last 10 minutes.
    SUM(vdrop) as vdrop, // Collect statistics on the number of video packets dropped in the last 10 minutes.
    SUM(alat) as alat, // Collect statistics on the audio latency in the last 10 minutes.
    SUM(vlat) as vlat, // Collect statistics on the video latency in the last 10 minutes.
FROM
    view_app_heartbeat_stream
GROUP BY
    TUMBLE(app_ts, INTERVAL '10' MINUTE), roomid;
```

- Latency statistics collected by region
  Collect statistics on the average end-to-end latency of audio and video data by region every 10 minutes.

```
CREATE VIEW region_lat_statistics_10min AS
SELECT
    CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
    region,
    SUM(alat)/COUNT(alat) as alat,
    SUM(vlat)/COUNT(vlat) as vlat,
FROM
    view_app_heartbeat_stream
GROUP BY
    TUMBLE(app_ts, INTERVAL '10' MINUTE), region;
```

- Overall frame freezing rate collected in real time
  Calculate the overall frame freezing rate by using the following formula: Number of online users who encounter frame freezing/Total number of online users × 100%. This metric can be used to measure the scope of users who encounter frame freezing.

```
CREATE VIEW block_total_statistics_10min AS
SELECT
    CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
    SUM(IF(ublock <> 0 OR dblock <> 0, 1, 0)) / CAST(COUNT(DISTINCT userid) AS DOUBLE
) as block_rate, // COUNT(DISTINCT) is supported only in Blink of a version later tha
n 1.4.4.
FROM
    view_app_heartbeat_stream
GROUP BY
    TUMBLE(app_ts, INTERVAL '10' MINUTE);
```

- Number of frame freezing times per user
  Calculate the number of frame freezing times per user by using the following formula: Total number of online frame freezing times/Total number of online users. This metric can be used to measure the overall severity of frame freezing based on the number of times frame freezing occurs.

```
CREATE VIEW block_peruser_statistics_10min AS
SELECT
    CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
    SUM(ublock+dblock) / CAST(COUNT(DISTINCT userid) AS DOUBLE) as block_peruser, //
COUNT(DISTINCT) is supported only in Blink of a version later than 1.4.4.
FROM
    view_app_heartbeat_stream
GROUP BY
    TUMBLE(app_ts, INTERVAL '10' MINUTE);
```

Demo code and source code

Alibaba Cloud provides a complete demo for you based on core video metric monitoring for live streaming described in this topic. You can refer to the demo code to register upstream and downstream storage data and to develop your own solution to monitor core video metrics. You can click Demo to download it. To use the demo, you must upload a CSV file as a source table to DataHub and create an ApsaraDB RDS result table to store the result data.

8.2.2. Digital operations for live streaming

This topic provides a use case to describe how to use Realtime Compute for Apache Flink to implement digital operations for live streaming.

Digital operations

This topic focuses on digital operations. You can use Realtime Compute for Apache Flink to monitor the operating status of streaming channels, such as popular videos and user trends, at your live streaming website in real time.

Solutions

- Business objectives
  - Collect statistics on the total number of users and user trend at your website.
  - Collect statistics on the total number of users and user trend of a channel.
  - Collect statistics on the top 10 popular channels at your website, and top 10 popular channels in each category.
- Data format

  Use the logs tracked in the client app as the raw data to collect statistics.

  The following table describes the data format of the tracked logs that the client app sends to the server.

Field name	Description
ip	The IP address of the client.
agent	The type of the client.
roomid	The ID of the channel.
userid	The ID of the user.
abytes	The audio bitrate.
afcnt	The number of audio frames.
adrop	The number of dropped audio frames.
afts	The audio timestamp.
alat	The end-to-end latency of audio frames.
vbytes	The video bitrate.
vfcnt	The number of video frames.
vdrop	The number of dropped video frames.
vfts	The video timestamp.
vlat	The end-to-end latency of video frames.
ublock	The number of upstream frame freezing times.
dblock	The number of downstream frame freezing times.
timestamp	The timestamp when a log was generated.
region	The region where live streaming is performed.

Log Service uses semi-structured storage and displays the preceding fields in the following log format:

```
{
    "ip": "ip",
    "agent": "agent",
    "roomid": "123456789",
    "userid": "123456789",
    "abytes": "123456",
    "afcnt": "34",
    "adrop": "3",
    "afts": "1515922566",
    "alat": "123",
    "vbytes": "123456",
    "vfcnt": "34",
    "vdrop": "4",
    "vfts": "1515922566",
    "vlat": "123",
    "ublock": "1",
    "dblock": "2",
        "timestamp": "1515922566",
    "region": "hangzhou"
}
```

- SQL statements
  - Collect statistics on the total number of users and user trend at your website.
    Use a new window every minute to collect statistics on the user trend at your website. The statistical result of the last minute in the trend is the current total number of users at your website.

    ```
    CREATE VIEW view_app_total_visit_1min AS
    SELECT
        CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
        COUNT(DISTINCT userid) as app_total_user_cnt
    FROM
        view_app_heartbeat_stream
    GROUP BY
        TUMBLE(app_ts, INTERVAL '1' MINUTE);
    ```

  - Collect statistics on the total number of users and user trend of a channel.
    Similarly, use a new window every minute to collect statistics on the user trend of a channel. The statistical result of the last minute in the trend is the current total number of users of the channel.

    ```
    CREATE VIEW view_app_room_visit_1min AS
    SELECT
        CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
        roomid as room_id,
        COUNT(DISTINCT userid) as app_room_user_cnt
    FROM
        view_app_heartbeat_stream
    GROUP BY
        TUMBLE(app_ts, INTERVAL '1' MINUTE), roomid;
    ```

- Rank the top 10 popular channels at your website.
  Collect statistics on the top 10 popular channels and promote these channels on the homepage to increase your website traffic and clicks.

```
CREATE VIEW view_app_room_visit_top10 AS
SELECT
  app_ts,
  room_id,
  app_room_user_cnt,
  rangking
FROM
(
    SELECT
        app_ts,
        room_id,
        app_room_user_cnt,
        ROW_NUMBER() OVER (PARTITION BY 1 ORDER BY app_room_user_cnt desc) AS
ranking
    FROM
        view_app_room_visit_1min
) WHERE ranking <= 10;
```

- Rank the top 10 popular channels in each category.

  To build a highly interactive social community and cover more live streaming scenarios to realize more profit, a platform operator can establish diversified channels at their live streaming website to meet the requirements of different user groups. For example, Taobao Live covers multiple categories such as makeup, men's wear, automotive, and fitness.

  The category and channel relationship table is a mapping table that is stored in an ApsaraDB RDS database.

```
CREATE TABLE dim_category_room (
    id    BIGINT,
    category_id BIGINT,
    category_name VARCHAR,
    room_id    BIGINT
      PRIMARY KEY (room_id),
      PERIOD FOR SYSTEM_TIME --The identifier of the dimension table.
 ) WITH (
     type= 'rds',
     url = 'xxxx', --The URL of your database.
     tableName = 'xxx', /--The name of your table.
     userName = 'xxx', --Your username.
     password = 'xxx' --Your password.
 );
```

  Join the dim_category_room table based on the channel ID and compute the rankings of channels in each category.

```
CREATE VIEW view_app_room_visit_1min AS
SELECT
    CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
    roomid as room_id,
    COUNT(DISTINCT userid) as app_room_user_cnt
FROM
    view_app_heartbeat_stream
GROUP BY
    TUMBLE(app_ts, INTERVAL '1' MINUTE), roomid;

// Join the dim_category_room table based on the channel ID.
CREATE VIEW view_app_category_visit_1min AS
SELECT
    r.app_ts,
    r.room_id,
    d.category_id,
    d.category_name,
    r.app_room_user_cnt
FROM
    view_app_room_visit_1min r
JOIN
    dim_category_room d
ON
    r.room_id = d.room_id;


// Compute the rankings of channels in each category.
CREATE VIEW view_app_category_visit_top10 AS
SELECT
  app_ts,
  category_id,
  category_name,
  app_room_user_cnt,
  rangking
FROM
(
    SELECT
        app_ts,
        room_id,
        category_id,
        category_name,
        app_room_user_cnt,
        ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY app_room_user_cnt
desc) AS ranking
    FROM
        view_app_category_visit_1min
) WHERE ranking <= 10;
```

Demo code and source code

Alibaba Cloud provides a complete demo. You can refer to the demo code to register upstream and downstream storage data and to develop your own digital operations solution for live streaming. When you use the demo, take note of the following items for the upstream and downstream storage:

- Upload a CSV file as a source table to DataHub.
- Create an ApsaraDB RDS dimension table.
- Create an ApsaraDB RDS result table.

9.Agreements

9.1. Realtime Compute for Apache Flink Service Level Agreement

For more information about the Realtime Compute for Apache Flink Service Level Agreement (SLA), see Alibaba Cloud Realtime Compute for Apache Flink Service Level Agreement.