数据湖探索

Flink SQL 语法参考

文档版本 01

发布日期 2025-09-09

华为技术有限公司

版权所有 © 华为技术有限公司 2025。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

 版权所有 © 华为技术有限公司

目 录

1 Flink Opensource SQL 语法参考简介 1
2 Flink Opensource SQL1.15 语法参考 2
2.1 SQL 语法约束与定义 2
2.1.1 语法支持类型 2
2.1.2 保留关键字 2
2.1.3 DDL 语法定义 4
2.1.3.1 CREATE TABLE 语句 4
2.1.3.2 CREATE CATALOG 语句 6
2.1.3.3 CREATE DATABASE 语句 7
2.1.3.4 CREATE VIEW 语句 7
2.1.3.5 CREATE FUNCTION 语句 8
2.1.4 DML 语法定义 8
2.2 Flink OpenSource SQL1.15 语法概览 11
2.3 Flink OpenSource SQL 1.15 版本使用说明 12
2.4 Format 12
2.4.1 Format 概述 12
2.4.2 Avro Format 13
2.4.3 Canal Format 16
2.4.4 Confluent Avro Format 20
2.4.5 CSV Format 23
2.4.6 Debezium Format 26
2.4.7 JSON Format 33
2.4.8 Maxwell Format 37
2.4.9 Ogg Format 40
2.4.10 Orc Format 44
2.4.11 Parquet Format 46
2.4.12 Raw Format 48
2.5 Connector 列表 51
2.5.1 Connector 概述 51
2.5.2 BlackHole 52
2.5.3 ClickHouse 53
2.5.4 DataGen 59
2.5.5 Doris 62

 版权所有 © 华为技术有限公司

2.5.5.1 Doris Connector 概述 63
2.5.5.2 Doris 源表 63
2.5.5.3 Doris 结果表 66
2.5.5.4 Doris 维表 69
2.5.6 DWS 72
2.5.6.1 DWS Connector 概述 72
2.5.6.2 DWS 源表（不推荐使用） 73
2.5.6.3 DWS 结果表（不推荐使用） 77
2.5.6.4 DWS 维表（不推荐使用） 83
2.5.7 Elasticsearch 88
2.5.8 对象存储 OBS 95
2.5.8.1 对象存储 OBS 源表 95
2.5.8.2 对象存储 OBS 结果表 96
2.5.9 Hbase 100
2.5.9.1 Hbase 源表 101
2.5.9.2 Hbase 结果表 105
2.5.9.3 Hbase 维表 111
2.5.10 Hive 117
2.5.10.1 创建 Hive Catalog 117
2.5.10.2 Hive 方言 120
2.5.10.3 Hive 源表 121
2.5.10.4 Hive 结果表 126
2.5.10.5 Hive 维表 128
2.5.10.6 使用 Temporal join 关联维表的最新分区 129
2.5.10.7 使用 Temporal join 关联维表的最新版本 131
2.5.11 Hudi 133
2.5.11.1 Hudi 源表 133
2.5.11.2 Hudi 结果表 138
2.5.12 JDBC 145
2.5.13 Kafka 153
2.5.14 MySql CDC 169
2.5.15 Print 181
2.5.16 Redis 184
2.5.16.1 Redis 源表 184
2.5.16.2 Redis 结果表 190
2.5.16.3 Redis 维表 201
2.5.17 Upsert Kafka 207
2.6 数据操作语句 DML 214
2.6.1 SELECT 214
2.6.2 INSERT INTO 218
2.6.3 集合操作 219
2.6.4 窗口 220

 版权所有 © 华为技术有限公司

2.6.4.1 窗口函数 227
2.6.4.2 窗口聚合 234
2.6.4.3 窗口 Top-N 237
2.6.4.4 窗口去重 239
2.6.4.5 窗口关联 240
2.6.5 分组聚合 243
2.6.6 Over 聚合 245
2.6.7 JOIN 246
2.6.8 OrderBy & Limit 249
2.6.9 Top-N 249
2.6.10 去重 250
2.7 函数 251
2.7.1 自定义函数 251
2.7.2 自定义函数类型推导 254
2.7.3 自定义函数参数传递 255
2.7.4 内置函数 259
2.7.4.1 比较函数 259
2.7.4.2 逻辑函数 263
2.7.4.3 算术函数 264
2.7.4.4 字符串函数 266
2.7.4.5 时间函数 271
2.7.4.6 条件函数 296
2.7.4.7 类型转换函数 297
2.7.4.8 集合函数 300
2.7.4.9 JSON 函数 300
2.7.4.10 值构建函数 305
2.7.4.11 值获取函数 306
2.7.4.12 分组函数 306
2.7.4.13 Hash 函数 306
2.7.4.14 聚合函数 307
2.7.4.15 表值函数 308
2.7.4.15.1 string_split 308
3 Flink Opensource SQL1.12 语法参考 311
3.1 SQL 语法约束与定义 311
3.1.1 语法支持类型 311
3.1.2 语法定义 311
3.1.2.1 DDL 语法定义 311
3.1.2.1.1 CREATE TABLE 语句 311
3.1.2.1.2 CREATE VIEW 语句 314
3.1.2.1.3 CREATE FUNCTION 语句 314
3.1.2.2 DML 语法定义 315
3.2 Flink OpenSource SQL1.12 语法概览 317

 版权所有 © 华为技术有限公司

3.3 数据定义语句 DDL 318
3.3.1 创建源表 318
3.3.1.1 DataGen 源表 318
3.3.1.2 DWS 源表 321
3.3.1.3 Hbase 源表 325
3.3.1.4 JDBC 源表 330
3.3.1.5 Kafka 源表 335
3.3.1.6 MySQL CDC 源表 347
3.3.1.7 Postgres CDC 源表 351
3.3.1.8 Redis 源表 355
3.3.1.9 Upsert Kafka 源表 362
3.3.1.10 FileSystem 源表 366
3.3.2 创建结果表 367
3.3.2.1 BlackHole 结果表 367
3.3.2.2 ClickHouse 结果表 368
3.3.2.3 DWS 结果表 372
3.3.2.4 Elasticsearch 结果表 377
3.3.2.5 Hbase 结果表 384
3.3.2.6 JDBC 结果表 390
3.3.2.7 Kafka 结果表 394
3.3.2.8 Print 结果表 404
3.3.2.9 Redis 结果表 406
3.3.2.10 Upsert Kafka 结果表 416
3.3.2.11 FileSystem 结果表 420
3.3.3 创建维表 424
3.3.3.1 DWS 维表 424
3.3.3.2 Hbase 维表 429
3.3.3.3 JDBC 维表 434
3.3.3.4 Redis 维表 439
3.3.4 Format 445
3.3.4.1 Avro Format 445
3.3.4.2 Canal Format 448
3.3.4.3 Confluent Avro Format 451
3.3.4.4 CSV Format 453
3.3.4.5 Debezium Format 455
3.3.4.6 JSON Format 457
3.3.4.7 Maxwell Format 461
3.3.4.8 Raw Format 463
3.4 数据操作语句 DML 465
3.4.1 SELECT 465
3.4.2 集合操作 468
3.4.3 窗口 469

 版权所有 © 华为技术有限公司

3.4.4 JOIN 476
3.4.5 OrderBy & Limit 479
3.4.6 Top-N 480
3.4.7 去重 481
3.5 函数 481
3.5.1 自定义函数 481
3.5.2 自定义函数类型推导 485
3.5.3 自定义函数参数传递 486
3.5.4 内置函数 489
3.5.4.1 数学运算函数 490
3.5.4.2 字符串函数 496
3.5.4.3 时间函数 503
3.5.4.4 条件函数 525
3.5.4.5 类型转换函数 526
3.5.4.6 集合函数 528
3.5.4.7 值构建函数 529
3.5.4.8 属性访问函数 529
3.5.4.9 Hash 函数 530
3.5.4.10 聚合函数 530
3.5.4.11 表值函数 531
3.5.4.11.1 string_split 531
4 Flink Opensource SQL1.10 语法参考 534
4.1 SQL 语法约束与定义 534
4.1.1 语法支持类型 534
4.1.2 语法定义 534
4.1.2.1 DDL 语法定义 534
4.1.2.1.1 CREATE TABLE 语句 534
4.1.2.1.2 CREATE VIEW 语句 537
4.1.2.1.3 CREATE FUNCTION 语句 537
4.1.2.2 DML 语法定义 538
4.2 Flink OpenSource SQL1.10 语法概览 540
4.3 数据定义语句 DDL 541
4.3.1 创建源表 541
4.3.1.1 Kafka 源表 541
4.3.1.2 DIS 源表 544
4.3.1.3 JDBC 源表 546
4.3.1.4 DWS 源表 548
4.3.1.5 Redis 源表 550
4.3.1.6 Hbase 源表 552
4.3.1.7 userDefined 源表 553
4.3.2 创建结果表 555
4.3.2.1 ClickHouse 结果表 555

 版权所有 © 华为技术有限公司

4.3.2.2 Kafka 结果表 558
4.3.2.3 Upsert Kafka 结果表 559
4.3.2.4 DIS 结果表 561
4.3.2.5 JDBC 结果表 562
4.3.2.6 DWS 结果表 564
4.3.2.7 Redis 结果表 567
4.3.2.8 SMN 结果表 570
4.3.2.9 Hbase 结果表 571
4.3.2.10 Elasticsearch 结果表 573
4.3.2.11 OpenTSDB 结果表 575
4.3.2.12 userDefined 结果表 577
4.3.2.13 Print 结果表 579
4.3.2.14 FileSytem 结果表 581
4.3.3 创建维表 583
4.3.3.1 创建 JDBC 维表 583
4.3.3.2 创建 DWS 维表 586
4.3.3.3 创建 Hbase 维表 588
4.4 数据操作语句 DML 590
4.4.1 SELECT 590
4.4.2 集合操作 593
4.4.3 窗口 594
4.4.4 JOIN 599
4.4.5 OrderBy & Limit 601
4.4.6 Top-N 602
4.4.7 去重 603
4.5 函数 604
4.5.1 自定义函数 604
4.5.2 内置函数 608
4.5.2.1 数学运算函数 608
4.5.2.2 字符串函数 614
4.5.2.3 时间函数 620
4.5.2.4 条件函数 642
4.5.2.5 类型转换函数 643
4.5.2.6 集合函数 645
4.5.2.7 值构建函数 646
4.5.2.8 属性访问函数 646
4.5.2.9 Hash 函数 646
4.5.2.10 聚合函数 647
4.5.2.11 表值函数 648
4.5.2.11.1 split_cursor 648
4.5.2.11.2 string_split 648

 版权所有 © 华为技术有限公司

1 Flink Opensource SQL 语法参考简介

表 1-1 Flink Opensource SQL 语法参考简介

Flink版本	说明	语法参考
Flink 1.15	• Flink 1.15版本在语法设计上实现了更高的兼容性，与主流开源技术标准保持一致。 • Flink 1.15版本新增读写Hive、Hudi等Connector。 • Flink 1.15版本数据同步迁移场景，优先推荐使用DataArts的数据集成。 • Flink 1.15版本支持集成DEW-CSMS凭证管理，提供隐私保护方案。 • Flink 1.15版本支持Flink Jar作业最小化提交。	**Flink OpenSource SQL1.15语法概览**
Flink 1.12	• Flink 1.12新增支持DataGen源表、DWS源表、JDBC源表、MySQL CDC源表、Postgres CDC源表、Redis源表、Upsert Kafka源表、Hbase源表。 • Flink 1.12新增支持小文件合并功能。 • Flink 1.12新增支持Redis维表、RDS维表。	**Flink OpenSource SQL1.12语法概览**
Flink 1.10	支持DIS源表结果表，支持DWS源表、结果表和维表，支持REDIS，支持自定义connector，支持TUMBLE窗口周期触发和控制延迟。	**Flink OpenSource SQL1.10语法概览**

版权所有 © 华为技术有限公司

2 Flink Opensource SQL1.15 语法参考

2.1 SQL 语法约束与定义

2.1.1 语法支持类型

DLI支持以下数据类型：

CHAR, VARCHAR, STRING, BOOLEAN, BINARY, VARBINARY, BYTES, DECIMAL, TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, DATE, TIME, TIMESTAMP, TIMESTAMP_LTZ, INTERVAL, ARRAY, MULTISET, MAP, ROW, RAW

在SQL语法中这些类型用于定义表中列的数据类型。

2.1.2 保留关键字

一些字符串的组合已经被预留为关键字以备未来使用。

如果使用以下字符串作为字段名，请在使用时使用反引号将该字段名包起来，例如 `value`, `count` 。

A, ABS, ABSOLUTE, ACTION, ADA, ADD, ADMIN, AFTER, ALL, ALLOCATE, ALLOW, ALTER, ALWAYS, AND, ANY, ARE, ARRAY, AS, ASC, ASENSITIVE, ASSERTION, ASSIGNMENT, ASYMMETRIC, AT, ATOMIC, ATTRIBUTE, ATTRIBUTES, AUTHORIZATION, AVG, BEFORE, BEGIN, BERNOULLI, BETWEEN, BIGINT, BINARY, BIT, BLOB, BOOLEAN, BOTH, BREADTH, BY, BYTES, C, CALL, CALLED, CARDINALITY, CASCADE, CASCADED, CASE, CAST, CATALOG, CATALOG_NAME, CEIL, CEILING, CENTURY, CHAIN, CHAR, CHARACTER, CHARACTERISTICS, CHARACTERS, CHARACTER_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_NAME, CHARACTER_SET_SCHEMA, CHAR_LENGTH, CHECK, CLASS_ORIGIN, CLOB, CLOSE, COALESCE, COBOL, COLLATE, COLLATION, COLLATION_CATALOG, COLLATION_NAME, COLLATION_SCHEMA, COLLECT, COLUMN, COLUMNS, COLUMN_NAME, COMMAND_FUNCTION, COMMAND_FUNCTION_CODE, COMMIT, COMMITTED, CONDITION, CONDITION_NUMBER, CONNECT, CONNECTION, CONNECTION_NAME, CONSTRAINT, CONSTRAINTS, CONSTRAINT_CATALOG, CONSTRAINT_NAME, CONSTRAINT_SCHEMA, CONSTRUCTOR, CONTAINS, CONTINUE, CONVERT, CORR, CORRESPONDING, COUNT, COVAR_POP, COVAR_SAMP, CREATE, CROSS, CUBE, CUME_DIST,

 版权所有 © 华为技术有限公司

CURRENT, CURRENT_CATALOG, CURRENT_DATE,
CURRENT_DEFAULT_TRANSFORM_GROUP, CURRENT_PATH, CURRENT_ROLE,
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP,
CURRENT_TRANSFORM_GROUP_FOR_TYPE, CURRENT_USER, CURSOR,
CURSOR_NAME, CYCLE, DATA, DATABASE, DATE, DATETIME_INTERVAL_CODE,
DATETIME_INTERVAL_PRECISION, DAY, DEALLOCATE, DEC, DECADE, DECIMAL,
DECLARE, DEFAULT, DEFAULTS, DEFERRABLE, DEFERRED, DEFINED, DEFINER,
DEGREE, DELETE, DENSE_RANK, DEPTH, DEREF, DERIVED, DESC, DESCRIBE,
DESCRIPTION, DESCRIPTOR, DETERMINISTIC, DIAGNOSTICS, DISALLOW,
DISCONNECT, DISPATCH, DISTINCT, DOMAIN, DOUBLE, DOW, DOY, DROP,
DYNAMIC, DYNAMIC_FUNCTION, DYNAMIC_FUNCTION_CODE, EACH, ELEMENT,
ELSE, END, END-EXEC, EPOCH, EQUALS, ESCAPE, EVERY, EXCEPT, EXCEPTION,
EXCLUDE, EXCLUDING, EXEC, EXECUTE, EXISTS, EXP, EXPLAIN, EXTEND, EXTERNAL,
EXTRACT, FALSE, FETCH, FILTER, FINAL, FIRST, FIRST_VALUE, FLOAT, FLOOR,
FOLLOWING, FOR, FOREIGN, FORTRAN, FOUND, FRAC_SECOND, FREE, FROM,
FULL, FUNCTION, FUSION, G, GENERAL, GENERATED, GET, GLOBAL, GO, GOTO,
GRANT, GRANTED, GROUP, GROUPING, HAVING, HIERARCHY, HOLD, HOUR,
IDENTITY, IMMEDIATE, IMPLEMENTATION, IMPORT, IN, INCLUDING, INCREMENT,
INDICATOR, INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INSTANCE,
INSTANTIABLE, INT, INTEGER, INTERSECT, INTERSECTION, INTERVAL, INTO,
INVOKER, IS, ISOLATION, JAVA, JOIN, K, KEY, KEY_MEMBER, KEY_TYPE, LABEL,
LANGUAGE, LARGE, LAST, LAST_VALUE, LATERAL, LEADING, LEFT, LENGTH, LEVEL,
LIBRARY, LIKE, LIMIT, LN, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR,
LOWER, M, MAP, MATCH, MATCHED, MAX, MAXVALUE, MEMBER, MERGE,
MESSAGE_LENGTH, MESSAGE_OCTET_LENGTH, MESSAGE_TEXT, METHOD,
MICROSECOND, MILLENNIUM, MIN, MINUTE, MINVALUE, MOD, MODIFIES,
MODULE, MODULES, MONTH, MORE, MULTISET, MUMPS, NAME, NAMES,
NATIONAL, NATURAL, NCHAR, NCLOB, NESTING, NEW, NEXT, NO, NONE,
NORMALIZE, NORMALIZED, NOT, NULL, NULLABLE, NULLIF, NULLS, NUMBER,
NUMERIC, OBJECT, OCTETS, OCTET_LENGTH, OF, OFFSET, OLD, ON, ONLY, OPEN,
OPTION, OPTIONS, OR, ORDER, ORDERING, ORDINALITY, OTHERS, OUT, OUTER,
OUTPUT, OVER, OVERLAPS, OVERLAY, OVERRIDING, PAD, PARAMETER,
PARAMETER_MODE, PARAMETER_NAME, PARAMETER_ORDINAL_POSITION,
PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_NAME,
PARAMETER_SPECIFIC_SCHEMA, PARTIAL, PARTITION, PASCAL, PASSTHROUGH,
PATH, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK, PLACING, PLAN,
PLI, POSITION, POWER, PRECEDING, PRECISION, PREPARE, PRESERVE, PRIMARY,
PRIOR, PRIVILEGES, PROCEDURE, PUBLIC, QUARTER, RANGE, RANK, RAW, READ,
READS, REAL, RECURSIVE, REF, REFERENCES, REFERENCING, REGR_AVGX,
REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX,
REGR_SXY, REGR_SYY, RELATIVE, RELEASE, REPEATABLE, RESET, RESTART,
RESTRICT, RESULT, RETURN, RETURNED_CARDINALITY, RETURNED_LENGTH,
RETURNED_OCTET_LENGTH, RETURNED_SQLSTATE, RETURNS, REVOKE, RIGHT,
ROLE, ROLLBACK, ROLLUP, ROUTINE, ROUTINE_CATALOG, ROUTINE_NAME,
ROUTINE_SCHEMA, ROW, ROWS, ROW_COUNT, ROW_NUMBER, SAVEPOINT,
SCALE, SCHEMA, SCHEMA_NAME, SCOPE, SCOPE_CATALOGS, SCOPE_NAME,
SCOPE_SCHEMA, SCROLL, SEARCH, SECOND, SECTION, SECURITY, SELECT, SELF,
SENSITIVE, SEQUENCE, SERIALIZABLE, SERVER, SERVER_NAME, SESSION,
SESSION_USER, SET, SETS, SIMILAR, SIMPLE, SIZE, SMALLINT, SOME, SOURCE,
SPACE, SPECIFIC, SPECIFICTYPE, SPECIFIC_NAME, SQL, SQLEXCEPTION, SQLSTATE,
SQLWARNING, SQL_TSI_DAY, SQL_TSI_FRAC_SECOND, SQL_TSI_HOUR,
SQL_TSI_MICROSECOND, SQL_TSI_MINUTE, SQL_TSI_MONTH, SQL_TSI_QUARTER,
SQL_TSI_SECOND, SQL_TSI_WEEK, SQL_TSI_YEAR, SQRT, START, STATE,

 版权所有 © 华为技术有限公司

STATEMENT, STATIC, STDDEV_POP, STDDEV_SAMP, STREAM, STRING, STRUCTURE, STYLE, SUBCLASS_ORIGIN, SUBMULTISET, SUBSTITUTE, SUBSTRING, SUM, SYMMETRIC, SYSTEM, SYSTEM_USER, TABLE, TABLESAMPLE, TABLE_NAME, TEMPORARY, THEN, TIES, TIME, TIMESTAMP, TIMESTAMPADD, TIMESTAMPDIFF, TIMEZONE_HOUR, TIMEZONE_MINUTE, TINYINT, TO, TOP_LEVEL_COUNT, TRAILING, TRANSACTION, TRANSACTIONS_ACTIVE, TRANSACTIONS_COMMITTED, TRANSACTIONS_ROLLED_BACK, TRANSFORM, TRANSFORMS, TRANSLATE, TRANSLATION, TREAT, TRIGGER, TRIGGER_CATALOG, TRIGGER_NAME, TRIGGER_SCHEMA, TRIM, TRUE, TYPE, UESCAPE, UNBOUNDED, UNCOMMITTED, UNDER, UNION, UNIQUE, UNKNOWN, UNNAMED, UNNEST, UPDATE, UPPER, UPSERT, USAGE, USER, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_CODE, USER_DEFINED_TYPE_NAME, USER_DEFINED_TYPE_SCHEMA, USING, VALUE, VALUES, VARBINARY, VARCHAR, VARYING, VAR_POP, VAR_SAMP, VERSION, VIEW, WEEK, WHEN, WHENEVER, WHERE, WIDTH_BUCKET, WINDOW, WITH, WITHIN, WITHOUT, WORK, WRAPPER, WRITE, XML, YEAR, ZONE

2.1.3 DDL 语法定义

2.1.3.1 CREATE TABLE 语句

功能描述

根据指定的表名创建一个表，如果同名表已经在 catalog 中存在了，则无法注册。

语法定义

```
CREATE TABLE [IF NOT EXISTS] [catalog_name.][db_name.]table_name
  (
    { <physical_column_definition> | <metadata_column_definition> | <computed_column_definition> }[ , ...n]
    [ <watermark_definition> ]
    [ <table_constraint> ][ , ...n]
  )
  [COMMENT table_comment]
  [PARTITIONED BY (partition_column_name1, partition_column_name2, ...)]
  WITH (key1=val1, key2=val2, ...)
  [ LIKE source_table [( <like_options> )] ]

<physical_column_definition>:
  column_name column_type [ <column_constraint> ] [COMMENT column_comment]

<column_constraint>:
  [CONSTRAINT constraint_name] PRIMARY KEY NOT ENFORCED

<table_constraint>:
  [CONSTRAINT constraint_name] PRIMARY KEY (column_name, ...) NOT ENFORCED

<metadata_column_definition>:
  column_name column_type METADATA [ FROM metadata_key ] [ VIRTUAL ]

<computed_column_definition>:
  column_name AS computed_column_expression [COMMENT column_comment]

<watermark_definition>:
  WATERMARK FOR rowtime_column_name AS watermark_strategy_expression

<source_table>:
  [catalog_name.][db_name.]table_name

<like_options>:
```

 版权所有 © 华为技术有限公司

```
{
  { INCLUDING | EXCLUDING } { ALL | CONSTRAINTS | PARTITIONS }
 | { INCLUDING | EXCLUDING | OVERWRITING } { GENERATED | OPTIONS | WATERMARKS }
}[, ...]
```

语法说明

COMPUTED COLUMN

计算列是一个使用 “column_name AS computed_column_expression” 语法生成的虚拟列。它由使用同一表中其他列的非查询表达式生成，并且不会在表中进行物理存储。例如，一个计算列可以使用 cost AS price * quantity 进行定义，这个表达式可以包含物理列、常量、函数或变量的任意组合，但这个表达式不能存在任何子查询。

在 Flink 中计算列一般用于为 CREATE TABLE 语句定义 时间属性。 处理时间属性 可以简单地通过使用了系统函数 PROCTIME() 的 proc AS PROCTIME() 语句进行定义。另一方面，由于事件时间列可能需要从现有的字段中获得，因此计算列可用于获得事件时间列。例如，原始字段的类型不是 TIMESTAMP(3) 或嵌套在 JSON 字符串中。

注意：

- 定义在一个数据源表（ source table ）上的计算列会在从数据源读取数据后被计算，它们可以在 SELECT 查询语句中使用。
- 计算列不可以作为 INSERT 语句的目标，在 INSERT 语句中，SELECT 语句的 schema 需要与目标表不带有计算列的 schema 一致。

WATERMARK

WATERMARK 定义了表的事件时间属性，其形式为 WATERMARK FOR rowtime_column_name AS watermark_strategy_expression 。

rowtime_column_name 把一个现有的列定义为一个为表标记事件时间的属性。该列的类型必须为 TIMESTAMP(3)，且是 schema 中的顶层列，它也可以是一个计算列。

watermark_strategy_expression 定义了 watermark 的生成策略。它允许使用包括计算列在内的任意非查询表达式来计算 watermark ； 表达式的返回类型必须是 TIMESTAMP(3)，表示了从 Epoch 以来的经过的时间。 返回的 watermark 只有当其不为空且其值大于之前发出的本地 watermark 时才会被发出（以保证 watermark 递增）。每条记录的 watermark 生成表达式计算都会由框架完成。 框架会定期发出所生成的最大的 watermark ，如果当前 watermark 仍然与前一个 watermark 相同、为空、或返回的 watermark 的值小于最后一个发出的 watermark ，则新的 watermark 不会被发出。 Watermark 根据 pipeline.auto-watermark-interval 中所配置的间隔发出。 如果 watermark 的间隔是 0ms ，那么每条记录都会产生一个 watermark，且 watermark 会在不为空并大于上一个发出的 watermark 时发出。

使用事件时间语义时，表必须包含事件时间属性和 watermark 策略。

Flink 提供了几种常用的 watermark 策略。

- 严格递增时间戳： WATERMARK FOR rowtime_column AS rowtime_column。

  发出到目前为止已观察到的最大时间戳的 watermark ，时间戳大于最大时间戳的行被认为没有迟到。
- 递增时间戳： WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '0.001' SECOND。

  发出到目前为止已观察到的最大时间戳减 1 的 watermark ，时间戳大于或等于最大时间戳的行被认为没有迟到。

 版权所有 © 华为技术有限公司

- 有界乱序时间戳： WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL 'string' timeUnit。

  发出到目前为止已观察到的最大时间戳减去指定延迟的 watermark ，例如，WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '5' SECOND 是一个 5 秒延迟的 watermark 策略。

```
CREATE TABLE Orders (
    user BIGINT,
    product STRING,
    order_time TIMESTAMP(3),
    WATERMARK FOR order_time AS order_time - INTERVAL '5' SECOND
) WITH ( . . . );
```

PRIMARY KEY

主键用作 Flink 优化的一种提示信息。主键限制表明一张表或视图的某个（些）列是唯一的并且不包含 Null 值。 主键声明的列都是非 nullable 的。因此主键可以被用作表行级别的唯一标识。

主键可以和列的定义一起声明，也可以独立声明为表的限制属性，不管是哪种方式，主键都不可以重复定义，否则 Flink 会报错。

有效性检查

SQL 标准主键限制可以有两种模式：ENFORCED 或者 NOT ENFORCED。 它申明了是否输入/出数据会做合法性检查（是否唯一）。Flink 不存储数据因此只支持 NOT ENFORCED 模式，即不做检查，用户需要自己保证唯一性。

Flink 假设声明了主键的列都是不包含 Null 值的，Connector 在处理数据时需要自己保证语义正确。

注意: 在 CREATE TABLE 语句中，创建主键会修改列的 nullable 属性，主键声明的列默认都是非 Nullable 的。

PARTITIONED BY

根据指定的列对已经创建的表进行分区。如果表使用 filesystem sink ，则将会为每个分区创建一个目录。

WITH OPTIONS

表属性用于创建 table source/sink ，一般用于寻找和创建底层的连接器。

表达式 key1=val1 的键和值必须为字符串文本常量。

注意： 表名可以为以下三种格式 1. catalog_name.db_name.table_name 2. db_name.table_name 3. table_name。使用catalog_name.db_name.table_name 的表将会与名为 “catalog_name” 的 catalog 和名为 “db_name” 的数据库一起注册到 metastore 中。使用 db_name.table_name 的表将会被注册到当前执行的 table environment 中的 catalog 且数据库会被命名为 “db_name”；对于 table_name, 数据表将会被注册到当前正在运行的catalog和数据库中。

注意： 使用 CREATE TABLE 语句注册的表均可用作 table source 和 table sink。 在被 DML 语句引用前，我们无法决定其实际用于 source 抑或是 sink。

2.1.3.2 CREATE CATALOG 语句

功能描述

根据给定的属性创建catalog。如果已经存在同名catalog会抛出异常。

 版权所有 © 华为技术有限公司

语法定义

```
CREATE CATALOG catalog_name
  WITH (key1=val1, key2=val2, ...)
```

语法说明

WITH OPTIONS

catalog属性一般用于存储关于这个catalog额外的信息。

表达式 key1=val1 中的键和值都是字符串文本常量。

2.1.3.3 CREATE DATABASE 语句

功能描述

根据给定的表属性创建数据库。如果数据库中已存在同名表会抛出异常。

语法定义

```
CREATE DATABASE [IF NOT EXISTS] [catalog_name.]db_name
  [COMMENT database_comment]
  WITH (key1=val1, key2=val2, ...)
```

语法说明

IF NOT EXISTS

如果数据库已经存在，则不会进行任何操作。

WITH OPTIONS

数据库属性一般用于存储关于这个数据库额外的信息。

表达式 key1=val1中的键和值都是字符串文本常量。

2.1.3.4 CREATE VIEW 语句

功能描述

根据给定的 query 语句创建一个视图。如果数据库中已经存在同名视图会抛出异常。

语法定义

```
CREATE [TEMPORARY] VIEW [IF NOT EXISTS] [catalog_name.][db_name.]view_name
  [{columnName [, columnName ]* }] [COMMENT view_comment]
  AS query_expression
```

语法说明

TEMPORARY

创建一个有 catalog 和数据库命名空间的临时视图，并覆盖原有的视图。

IF NOT EXISTS

如果该视图已经存在，则不会进行任何操作。

 版权所有 © 华为技术有限公司

示例

创建一个名为viewName的视图。

```
create view viewName as select * from dataSource
```

2.1.3.5 CREATE FUNCTION 语句

功能描述

创建一个有 catalog 和数据库命名空间的 catalog function ，需要指定一个 identifier ，可指定 language tag 。 若catalog 中，已经有同名的函数注册了，则无法注册。如果 language tag 是 JAVA 或者 SCALA ，则 identifier 是 UDF 实现类的全限定名。

如果您需要了解创建自定义函数的步骤请参考**自定义函数**。

语法定义

```
CREATE [TEMPORARY|TEMPORARY SYSTEM] FUNCTION
  [IF NOT EXISTS] [[catalog_name.]db_name.]function_name
  AS identifier [LANGUAGE JAVA|SCALA]
```

语法说明

TEMPORARY

创建一个有 catalog 和数据库命名空间的临时 catalog function ，并覆盖原有的 catalog function 。

TEMPORARY SYSTEM

创建一个没有数据库命名空间的临时系统 catalog function ，并覆盖系统内置的函数。

IF NOT EXISTS

如果该函数已经存在，则不会进行任何操作。

LANGUAGE JAVA|SCALA

Language tag 用于指定 Flink runtime 如何执行这个函数。目前，只支持 JAVA, SCALA，且函数的默认语言为 JAVA。

示例

创建一个名为STRINGBACK的函数。

```
create function STRINGBACK as 'com.dli.StringBack'
```

2.1.4 DML 语法定义

约束限制

- Flink SQL 对于标识符（表、属性、函数名）有类似于 Java 的词法约定:
  - 不管是否引用标识符，都保留标识符的大小写。
  - 且标识符需区分大小写。

 版权所有 © 华为技术有限公司

  - 与 Java 不一样的地方在于，通过反引号，可以允许标识符带有非字母的字符（如："SELECT a AS `my field` FROM t"）。
- 字符串文本常量需要被单引号包起来（如 SELECT 'Hello World' ）。两个单引号表示转义（如 SELECT 'It''s me.'）。字符串文本常量支持 Unicode 字符，如需明确使用 Unicode 编码，请使用以下语法：
  - 使用反斜杠（\）作为转义字符（默认）：SELECT U&'\263A'
  - 使用自定义的转义字符： SELECT U&'#263A' UESCAPE '#'

语法定义

```
INSERT INTO table_name [PARTITION part_spec] query

part_spec:  (part_col_name1=val1 [, part_col_name2=val2, ...])

query:
  values
  | {
      select
      | selectWithoutFrom
      | query UNION [ ALL ] query
      | query EXCEPT query
      | query INTERSECT query
    }
    [ ORDER BY orderItem [, orderItem ]* ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start { ROW | ROWS } ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]

orderItem:
  expression [ ASC | DESC ]

select:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
  [ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]

selectWithoutFrom:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }

projectItem:
  expression [ [ AS ] columnAlias ]
  | tableAlias . *

tableExpression:
  tableReference [, tableReference ]*
  | tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ] JOIN tableExpression [ joinCondition ]

joinCondition:
  ON booleanExpression
  | USING '(' column [, column ]* ')'

tableReference:
  tablePrimary
  [ matchRecognize ]
  [ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ] ]

tablePrimary:
  [ TABLE ] [ [ catalogName . ] schemaName . ] tableName
  | LATERAL TABLE '(' functionName '(' expression [, expression ]* ')' ')'
  | UNNEST '(' expression ')'
```

 版权所有 © 华为技术有限公司

```
values:
  VALUES expression [, expression ]*

groupItem:
  expression
  | '(' ')'
  | '(' expression [, expression ]* ')'
  | CUBE '(' expression [, expression ]* ')'
  | ROLLUP '(' expression [, expression ]* ')'
  | GROUPING SETS '(' groupItem [, groupItem ]* ')'

windowRef:
    windowName
  | windowSpec

windowSpec:
    [ windowName ]
    '('
    [ ORDER BY orderItem [, orderItem ]* ]
    [ PARTITION BY expression [, expression ]* ]
    [
        RANGE numericOrIntervalExpression {PRECEDING}
      | ROWS numericExpression {PRECEDING}
    ]
    ')'

matchRecognize:
      MATCH_RECOGNIZE '('
      [ PARTITION BY expression [, expression ]* ]
      [ ORDER BY orderItem [, orderItem ]* ]
      [ MEASURES measureColumn [, measureColumn ]* ]
      [ ONE ROW PER MATCH ]
      [ AFTER MATCH
            ( SKIP TO NEXT ROW
            | SKIP PAST LAST ROW
            | SKIP TO FIRST variable
            | SKIP TO LAST variable
            | SKIP TO variable )
      ]
      PATTERN '(' pattern ')'
      [ WITHIN intervalLiteral ]
      DEFINE variable AS condition [, variable AS condition ]*
      ')'

measureColumn:
      expression AS alias

pattern:
      patternTerm [ '|' patternTerm ]*

patternTerm:
      patternFactor [ patternFactor ]*

patternFactor:
      variable [ patternQuantifier ]

patternQuantifier:
      '*'
  |   '*?'
  |   '+'
  |   '+?'
  |   '?'
  |   '??'
  |   '{' { [ minRepeat ], [ maxRepeat ] } '}' ['?']
  |   '{' repeat '}'
```

 版权所有 © 华为技术有限公司

2.2 Flink OpenSource SQL1.15 语法概览

本章节介绍目前DLI所提供的Flink OpenSource SQL1.15语法列表。参数说明，示例等详细信息请参考具体的语法说明。

创建表相关语法

表 2-1 创建表相关语法

语法分类	功能描述
Format	**Avro**
	Canal
	Confluent Avro
	CSV
	Debezium
	JSON
	Maxwell
	Ogg
	Orc
	Parquet
	Raw
Connectors	**BlackHole**
	ClickHouse
	DataGen
	Doris
	DWS
	Elasticsearch
	FileSystem
	Hbase
	Hive
	JDBC
	Kafka
	Print
	Redis

版权所有 © 华为技术有限公司

语法分类	功能描述
	Upsert Kafka

2.3 Flink OpenSource SQL 1.15 版本使用说明

如果您的作业是从Flink1.12版本切换至Flink 1.15，在使用Flink OpenSource SQL 1.15时请注意以下使用说明。

- Flink SQL采用SQL Client 提交方式，相比Flink1.12的优化参数，Flink 1.15需要在SQL脚本使用SET 'key'='value'；进行配置。详细语法请参考**SQL Client Configuration**。
- Flink 1.15新增Flink Connector列表如下：Doris Connector、Hive Connector。详细操作请参考**Flink OpenSource SQL1.15语法概览**。
- Flink 1.15需要配置自定义委托，并在作业中配置委托信息。委托的权限需要根据作业具体的业务场景需求进行配置。详情操作请参考**DLI自定义委托**。
- Flink 1.15作业管理凭据的方法：
  - Flink OpenSource SQL中推荐使用DEW管理密码密钥等访问凭据信息。请参考**Flink Opensource SQL使用DEW管理访问凭据**。
  - Flink Jar 作业使用固定AKSK访问OBS、Flink Jar 获取委托的临时AKSK、Flink SQL UDF 获取委托的临时AKSK场景。请参考**Flink作业委托场景开发指导**。
- Flink 1.15 Jar 读取用户自定义配置文件的方式相比Flink 1.12存在差异。详细操作说明请参考**使用Flink Jar写入数据到OBS开发指南**。
- Flink 1.15 Jar 程序，采用反向类加载机制（child-first），可通过优化参数设置某些依赖包由父类加载器加载：parent.first.classloader.jars=test1.jar,test2.jar
- Flink 1.15 Jar 系统内置jar包清单，请在FLink作业的日志中获取Flink 1.15相关依赖包信息：
  a. 查看Flink日志。
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 单击作业名称，选择“运行日志”。
     iii. 控制台只展示最新的运行日志，更多日志信息请查看保存日志的OBS桶。
  b. 在日志中搜索依赖包信息。
     在日志中搜索“Classpath:”即可查看相关依赖包信息。
- Flink 1.15 不再支持DLI程序包管理的功能，依赖包、依赖文件等请在编辑作业时直接选择上传的OBS路径。

2.4 Format

2.4.1 Format 概述

Flink 提供了一套与表连接器（table connector）一起使用的表格式（table format）。

版权所有 © 华为技术有限公司

表格式是一种存储格式，定义了如何把二进制数据映射到表的列上。

表 2-2 Flink 支持格式

Formats	支持的Connectors
CSV	Kafka, Upsert Kafka, FileSystem
JSON	Kafka, Upsert Kafka, FileSystem, Elasticsearch
Avro	Kafka, Upsert Kafka, FileSystem
Confluent Avro	Kafka, Upsert Kafka
Debezium	Kafka, FileSystem
Canal	Kafka, FileSystem
Maxwell	Kafka, FileSystem
Ogg	Kafka, FileSystem
Orc	FileSystem
Parquet	FileSystem
Raw	Kafka, Upsert Kafka, FileSystem

2.4.2 Avro Format

功能描述

Avro格式允许基于Avro schema 读取和写入Avro 数据。目前，Avro schema 从表schema 推导。

更多具体使用可参考开源社区文档：Avro Format。

支持的 Connector

- Kafka
- Upsert Kafka
- FileSystem

参数说明

表 2-3 参数说明

参数	是否必选	默认值	类型	说明
format	是	（none）	String	指定使用格式，这里应该是'avro'。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
avro.codec	否	（none）	String	仅用于FileSystem，avro 压缩编解码器。默认 snappy 压缩。目前支持：null, deflate、snappy、bzip2、xz。

数据类型映射

目前，Avro schema 通常是从 table schema 中推导而来。尚不支持显式定义 Avro schema。因此，下表列出了从 Flink 类型到 Avro 类型的类型映射。

除了下面列出的类型，Flink 支持读取/写入 nullable 的类型。Flink 将 nullable 的类型映射到 Avro union(something, null)，其中 something 是从 Flink 类型转换的 Avro 类型。

表 2-4 数据类型映射

Flink SQL类型	Avro类型	Avro逻辑类型
CHAR / VARCHAR / STRING	string	-
BOOLEAN	boolean	-
BINARY / VARBINARY	bytes	-
DECIMAL	fixed	decimal
TINYINT	int	-
SMALLINT	int	-
INT	int	-
BIGINT	long	-
FLOAT	float	-
DOUBLE	double	-
DATE	int	date
TIME	int	time-millis
TIMESTAMP	long	timestamp-millis
ARRAY	array	-
MAP(key 必须是 string/char/varchar 类型)	map	-
MULTISET(元素必须是 string/char/varchar 类型)	map	-
ROW	record	-

版权所有 © 华为技术有限公司

示例

读取kafka中的数据，以avro格式反序列化，并输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.15，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'avro'
);


CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);
insert into printSink select * from kafkaSource;
```

步骤3 向kafka中以avro的序列化方式插入如下数据：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
+I[202103241000000001, webShop, 2021-03-24 10:00:00, 100.0, 100.0, 2021-03-24 10:02:03, 0001, Alice,
330106]
```

 版权所有 © 华为技术有限公司

```
+I[202103241606060001, appShop, 2021-03-24 16:06:06, 200.0, 180.0, 2021-03-24 16:10:06, 0001, Alice, 330106]
```

----结束

2.4.3 Canal Format

功能描述

Canal是一个 CDC（ChangeLog Data Capture，变更日志数据捕获）工具，可以实时地将 MySQL 变更传输到其他系统。Canal 为变更日志提供了统一的数据格式，并支持使用 JSON 或 protobuf序列化消息（Canal 默认使用 protobuf）。

Flink 支持将 Canal 的 JSON 消息解析为 INSERT / UPDATE / DELETE 消息到 Flink SQL 系统中。在很多情况下，利用这个特性非常的有用，例如

- 将增量数据从数据库同步到其他系统
- 日志审计
- 数据库的实时物化视图
- 关联维度数据库的变更历史，等等。

Flink 还支持将 Flink SQL 中的 INSERT / UPDATE / DELETE 消息编码为 Canal 格式的 JSON 消息，输出到 Kafka 等存储中。 但需要注意的是，目前 Flink 还不支持将 UPDATE_BEFORE 和 UPDATE_AFTER 合并为一条 UPDATE 消息。因此，Flink 将 UPDATE_BEFORE 和 UPDATE_AFTER 分别编码为 DELETE 和 INSERT 类型的 Canal 消息。

更多具体使用可参考开源社区文档：**Canal Format**。

支持的 Connector

- Kafka
- Filesystem

参数说明

表 2-5 参数说明

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定要使用的格式，此处应为 'canal-json'.
canal-json.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
canal-json.timestamp-format.standard	否	'SQL'	String	指定输入和输出时间戳格式。当前支持的值是：'SQL'和'ISO-8601'。 • 选项 'SQL' 将解析 "yyyy-MM-dd HH:mm:ss.s{precision}" 格式的输入时间戳，例如 '2020-12-30 12:13:14.123'，并以相同格式输出时间戳。 • 选项 'ISO-8601' 将解析 "yyyy-MM-ddTHH:mm:ss.s{precision}" 格式的输入时间戳，例如 '2020-12-30T12:13:14.123'，并以相同的格式输出时间戳。
canal-json.map-null-key.mode	否	'FALL'	String	指定处理 Map 中 key 值为空的方法. 当前支持的值有'FAIL', 'DROP'和'LITERAL'。 • Option 'FAIL' 将抛出异常，如果遇到 Map 中 key 值为空的数据。 • Option 'DROP' 将丢弃 Map 中 key 值为空的数据项。 • Option 'LITERAL' 将使用字符串常量来替换 Map 中的空 key 值。字符串常量的值由 'canal-json.map-null-key.literal' 定义。
canal-json.map-null-key.literal	否	'null'	String	当 'canal-json.map-null-key.mode' 是 LITERAL 的时候，指定字符串常量替换 Map 中的空 key 值。
canal-json.encode.decimal-as-plain-number	否	false	Boolean	将所有 DECIMAL 类型的数据保持原状，不使用科学计数法表示。例：0.000000027 默认会表示为 2.7E-8。当此选项设为 true 时，则会表示为 0.000000027。
canal-json.database.include	否	(none)	String	一个可选的正则表达式，通过正则匹配 Canal 记录中的 "database" 元字段，仅读取指定数据库的 changelog 记录。正则字符串与 Java 的 **Pattern** 兼容。
canal-json.table.include	否	(none)	String	一个可选的正则表达式，通过正则匹配 Canal 记录中的 "table" 元字段，仅读取指定表的 changelog 记录。正则字符串与 Java 的 **Pattern** 兼容。

版权所有 © 华为技术有限公司

元数据

元数据可以在 DDL 中作为只读（虚拟）meta 列声明。

Format的元数据只有在相应的连接器使用元数据时才可用。目前，只有Kafka连接器。

表 2-6 元数据

Key	数据类型	说明
database	STRING NULL	源数据库。对应于Canal记录中的database字段(如果可用)。
table	STRING NULL	源数据库表。对应于Canal中的table字段(如果可用)。
sql-type	MAP<STRING, INT> NULL	各种sql类型的Map。对应于Canal记录中的sqlType字段(如果可用)。
pk-names	ARRAY<STRING> NULL	主键名数组。对应于Canal记录中的pkNames字段(如果可用)。
ingestion-timestamp	TIMESTAMP_LTZ(3) NULL	connector处理事件的时间戳。对应Canal记录中的ts字段。

元数据的使用用例参考如下：

```
CREATE TABLE KafkaTable (
  origin_database STRING METADATA FROM 'value.database' VIRTUAL,
  origin_table STRING METADATA FROM 'value.table' VIRTUAL,
  origin_sql_type MAP<STRING, INT> METADATA FROM 'value.sql-type' VIRTUAL,
  origin_pk_names ARRAY<STRING> METADATA FROM 'value.pk-names' VIRTUAL,
  origin_ts TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL,
  user_id BIGINT,
  item_id BIGINT,
  behavior STRING
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'earliest-offset',
  'value.format' = 'canal-json'
);
```

示例

使用canal-json读取kafka中的canal记录，并输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列

 版权所有 © 华为技术有限公司

连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.15版本，并提交运行，其代码如下：

```
create table kafkaSource(
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
  ) with (
    'connector' = 'kafka',
    'topic' = '<yourTopic>',
    'properties.group.id' = '<yourGroupId>',
    'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'canal-json'
);
create table printSink(
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
   ) with (
     'connector' = 'print'
   );
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据，该数据表示MySQL products 表有4列（id，name，description 和 weight）。JSON 消息是 products 表上的一个更新事件，表示 id = 111 的行数据上 weight 字段值从5.15变更成为 5.18。

```
{
  "data": [
    {
      "id": "111",
      "name": "scooter",
      "description": "Big 2-wheel scooter",
      "weight": "5.18"
    }
  ],
  "database": "inventory",
  "es": 1589373560000,
  "id": 9,
  "isDdl": false,
  "mysqlType": {
    "id": "INTEGER",
    "name": "VARCHAR(255)",
    "description": "VARCHAR(512)",
    "weight": "FLOAT"
  },
  "old": [
    {
      "weight": "5.15"
    }
  ],
  "pkNames": [
    "id"
  ],
  "sql": "",
  "sqlType": {
    "id": 4,
    "name": 12,
    "description": 12,
    "weight": 7
  },
  "table": "products",
  "ts": 1589373560798,
  "type": "UPDATE"
}
```

 版权所有 © 华为技术有限公司

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
-U[111, scooter, Big 2-wheel scooter, 5.15]
+U[111, scooter, Big 2-wheel scooter, 5.18]
```

----结束

2.4.4 Confluent Avro Format

功能描述

Avro Schema Registry (avro-confluent) 格式能让您读取被 io.confluent.kafka.serializers.KafkaAvroSerializer 序列化的记录，以及可以写入成能被 io.confluent.kafka.serializers.KafkaAvroDeserializer 反序列化的记录。

当以这种格式读取（反序列化）记录时，将根据记录中编码的 schema 版本 id 从配置的 Confluent Schema Registry 中获取 Avro writer schema ，而从 table schema 中推断出 reader schema。

当以这种格式写入（序列化）记录时，Avro schema 是从 table schema 中推断出来的，并会用来检索要与数据一起编码的 schema id。我们会在配置的 Confluent Schema Registry 中配置的 **subject** 下，检索 schema id。subject 通过 avro-confluent.subject 参数来指定。

支持的 connector

- kafka
- upsert kafka

参数说明

表 2-7 参数说明

参数	是否必选	默认值	类型	说明
format	是	无	String	指定使用格式，此处使用'avro-confluent'。
avro-confluent.basic-auth.credentials-source	否	无	String	Schema Registry的基本身份验证凭据源。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
avro-confluent.basic-auth.user-info	否	无	String	Schema Registry的基本身份验证用户信息。
avro-confluent.bearer-auth.credentials-source	否	无	String	Schema Registry的承载身份验证凭据源。
avro-confluent.bearer-auth.token	否	无	String	Schema Registry的承载身份验证Token。
avro-confluent.properties	否	无	Map	转发到底层Schema Registry的属性Map。这对于没有通过Flink显示配置的配置项非常有用。但是，请注意，Flink配置项具有更高的优先级。
avro-confluent.ssl.keystore.location	否	无	String	SSL keystore的位置/文件。
avro-confluent.ssl.keystore.password	否	无	String	SSL keystore的密码。
avro-confluent.ssl.truststore.location	否	无	String	SSL truststore的位置/文件。
avro-confluent.ssl.truststore.password	否	无	String	SSL truststore的密码。
avro-confluent.subject	否	无	String	用于在序列化期间此格式使用的注册schema的Confluent Schema Registry主题。默认情况下，'kafka'和'upsert-kafka'连接器使用'<topic_name>-value'或'<topic_name>-key'作为默认主题名称，如果此格式用作键或值的格式。但是对于其他连接器（例如'filesystem'），在用作sink时需要使用主题选项。
avro-confluent.url	否	无	String	用于获取/注册架构的Confluent Schema Registry的URL。

版权所有 © 华为技术有限公司

数据类型映射

目前 Apache Flink 都是从 table schema 去推断反序列化期间的 Avro reader schema 和序列化期间的 Avro writer schema。显式地定义 Avro schema 暂不支持。 **Avro Format**中描述了 Flink 数据类型和 Avro 类型的对应关系。

除了此处列出的类型之外，Flink 还支持读取/写入可为空（ nullable ）的类型。 Flink 将可为空的类型映射到 Avro union(something, null), 其中 something 是从 Flink 类型转换的 Avro 类型。

示例

从kafka中作为source的topic中读取json数据，并以confluent avro的形式写入作为sink的topic中。

1. 根据kafka和ecs所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka和ecs的地址测试队列连通性（ 通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka或ecs的地址 > 测试 ）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。
2. 购买ecs集群，并下载5.5.2版本的**confluent**和jdk1.8.0_232，并上传到购买的ecs集群中，然后使用下述命令解压（ 假设解压目录分别为confluent-5.5.2和jdk1.8.0_232 ）。
   ```
   tar zxvf confluent-5.5.2-2.11.tar.gz
   tar zxvf jdk1.8.0_232.tar.gz
   ```
3. 使用下述命令在当前ecs集群中安装jdk1.8.0_232(其中<yourJdkPath>可以在jdk1.8.0_232文件夹下使用"pwd"查看)：
   ```
   export JAVA_HOME=<yourJdkPath>
   export PATH=$JAVA_HOME/bin:$PATH
   export CLASSPATH=.:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
   ```
4. 进入confluent-5.5.2/etc/schema-registry/目录下，修改schema-registry.properties文件中如下配置项：
   ```
   listeners=http://<yourEcsIp>:8081
   kafkastore.bootstrap.servers=<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>
   ```
5. 将ecs切换到confluent-5.5.2目录下，使用下述命令启动confluent：
   ```
   bin/schema-registry-start etc/schema-registry/schema-registry.properties
   ```
6. 创建flink opensource sql作业，选择版本flink 1.15，并选择保存日志，然后提交运行：
   ```
   CREATE TABLE kafkaSource (
     order_id string,
     order_channel string,
     order_time string,
     pay_amount double,
     real_pay double,
     pay_time string,
     user_id string,
     user_name string,
     area_id string
   ) WITH (
     'connector' = 'kafka',
     'topic' = 'kafkaSourceTopic',
     'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
     'properties.group.id' = 'GroupId',
     'scan.startup.mode' = 'latest-offset',
     'format' = 'json'
   );

   CREATE TABLE kafkaSink (
     order_id string,
   ```


版权所有 © 华为技术有限公司

```
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'kafkaSinkTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'format' = 'avro-confluent',
  'avro-confluent.url' = 'http://EcsIp:8081'
);
insert into kafkaSink select * from kafkaSource;
```

7. 向kafka中插入如下数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

8. 读取kafka的作为sink的topic的数据，则可发现数据已经写入，且schema已经保存到kafka的_schema的topic中。

2.4.5 CSV Format

功能描述

CSV Format 允许我们基于CSV schema 进行解析和生成CSV 数据。目前的CSV schema 是基于table schema 推导出来的。更多具体使用可参考开源社区文档：**CSV Format**。

支持的 Connector

- Kafka
- Upsert Kafka
- FileSystem

参数说明

表 2-8 参数说明

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定要使用的格式，这里应该是 'csv'。
csv.field-delimiter	否	,	String	字段分隔符 (默认',')，必须为单字符。您可以使用反斜杠字符指定一些特殊字符，例如 '\t' 代表制表符。 您也可以通过 unicode 编码在纯 SQL 文本中指定一些特殊字符，例如'csv.field-delimiter' = U&'\0001' 代表 0x01 字符。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
csv.disable-quote-character	否	false	Boolean	是否禁止对引用的值使用引号 (默认是 false). 如果禁止，选项 'csv.quote-character' 不能设置。
csv.quote-character	否	‘’	String	用于围住字段值的引号字符 (默认").
csv.allow-comments	否	false	Boolean	是否允许忽略注释行（默认不允许），注释行以 '#' 作为起始字符。 如果允许注释行，请确保 csv.ignore-parse-errors 也开启了从而允许空行。
csv.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。
csv.array-element-delimiter	否	;	String	分隔数组和行元素的字符串(默认';').
csv.escape-character	否	(none)	String	转义字符(默认关闭).
csv.null-literal	否	(none)	String	是否将 "null" 字符串转化为 null 值。

数据类型映射

目前 CSV 的 schema 都是从 table schema 推断而来的。显式地定义 CSV schema 暂不支持。 Flink 的 CSV Format 数据使用 **jackson databind API** 去解析 CSV 字符串。

表 2-9 数据类型映射

Flink SQL 类型	CSV 类型
CHAR / VARCHAR / STRING	string
BOOLEAN	boolean
BINARY / VARBINARY	string with encoding: base64
DECIMAL	number
TINYINT	number
SMALLINT	number
INT	number
BIGINT	number

版权所有 © 华为技术有限公司

Flink SQL 类型	CSV 类型
FLOAT	number
DOUBLE	number
DATE	string with format: date
TIME	string with format: time
TIMESTAMP	string with format: date-time
INTERVAL	number
ARRAY	array
ROW	object

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv'
);


CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);
insert into printSink select * from kafkaSource;
```

 版权所有 © 华为技术有限公司

步骤3 向kafka的作为source的topic中插入下列数据：

```
202103251505050001,appShop,2021-03-25 15:05:05,500.00,400.00,2021-03-25 15:10:00,0003,Cindy,330108

202103241606060001,appShop,2021-03-24 16:06:06,200.00,180.00,2021-03-24 16:10:06,0001,Alice,330106
```

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
+I[202103251505050001, appShop, 2021-03-25 15:05:05, 500.0, 400.0, 2021-03-25 15:10:00, 0003, Cindy, 330108]
+I[202103241606060001, appShop, 2021-03-24 16:06:06, 200.0, 180.0, 2021-03-24 16:10:06, 0001, Alice, 330106]
```

----结束

2.4.6 Debezium Format

功能描述

Debezium是一个 CDC（Changelog Data Capture，变更数据捕获）的工具，可以把其他数据库的更改实时流式传输到 Kafka 中。 Debezium 为变更日志提供了统一的格式结构，并支持使用 JSON 和 Apache Avro 序列化消息。

Flink 支持将 Debezium JSON 和 Avro 消息解析为 INSERT / UPDATE / DELETE 消息到 Flink SQL 系统中。在很多情况下，利用这个特性非常的有用，例如：

- 将增量数据从数据库同步到其他系统
- 日志审计
- 数据库的实时物化视图
- 关联维度数据库的变更历史

Flink 还支持将 Flink SQL 中的 INSERT / UPDATE / DELETE 消息编码为 Debezium 格式的 JSON 或 Avro 消息，输出到 Kafka 等存储中。 但需要注意的是，目前 Flink 还不支持将 UPDATE_BEFORE 和 UPDATE_AFTER 合并为一条 UPDATE 消息。因此，Flink 将 UPDATE_BEFORE 和 UPDATE_AFTER 分别编码为 DELETE 和 INSERT 类型的 Debezium 消息。

更多具体使用可参考开源社区文档：**Debezium Format**。

支持的 Connector

- Kafka
- Filesystem

注意事项

- 重复的变更事件

  在正常的操作环境下，Debezium 应用能以exactly-once的语义投递每条变更事件。在这种情况下，Flink 消费 Debezium 产生的变更事件能够工作得很好。 当发生故障时，Debezium应用只能保证at-least-once的投递语义。即在非正常情况

下，Debezium可能会投递重复的变更事件到Kafka中，当Flink从Kafka中消费的时候就会得到重复的事件。 这可能会导致Flink query的运行得到错误的结果或者非预期的异常。

解决方案：将作业参数 **table.exec.source.cdc-events-duplicate** 设置成true，并在该source上定义PRIMARY KEY。

框架会生成一个额外的有状态算子，使用该primary key来对变更事件去重并生成一个规范化的changelog流。

更新信息请参考**Debezium 官方文档**。

- 消费Debezium Postgres Connector产生的数据

  如果您正在使用Debezium PostgreSQL Connector捕获变更到 Kafka，请确保被监控表的**REPLICA IDENTITY** 已经被配置成FULL ，默认值是DEFAULT。 否则，Flink SQL将无法正确解析Debezium数据。

  当配置为 FULL 时，更新和删除事件将完整包含所有列的之前的值。

  当为其他配置时，更新和删除事件的“before”字段将只包含primary key字段的值，或者为 null（没有 primary key）。

  您可以通过运行 ALTER TABLE <your-table-name> REPLICA IDENTITY FULL 来更改 REPLICA IDENTITY 的配置。

参数说明

Flink 提供了 debezium-avro-confluent 和 debezium-json 两种 format 来解析 Debezium 生成的 JSON 格式和 Avro 格式的消息。 请使用 debezium-avro-confluent 来解析 Debezium 的 Avro 消息，使用 debezium-json 来解析 Debezium 的 JSON 消息。

表 2-10 Debezium Avro 参数说明

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定使用格式，此处使用'debezium-avro-confluent'。
debezium-avro-confluent.basic-auth.credentials-source	否	(none)	String	Schema Registry的基本身份验证凭据源。
debezium-avro-confluent.basic-auth.user-info	否	(none)	String	Schema Registry的基本身份验证用户信息。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
debezium-avro-confluent.bearer-auth.credentials-source	否	(none)	String	Schema Registry的承载身份验证凭据源。
debezium-avro-confluent.bearer-auth.token	否	(none)	String	Schema Registry的承载身份验证Token。
debezium-avro-confluent.properties	否	(none)	Map	转发到底层Schema Registry的属性Map。这对于没有通过Flink显示配置的配置项非常有用。但是，请注意，Flink配置项具有更高的优先级。
debezium-avro-confluent.ssl.keystore.location	否	(none)	String	SSL keystore的位置/文件。
debezium-avro-confluent.ssl.keystore.password	否	(none)	String	SSL keystore的密码。
debezium-avro-confluent.ssl.truststore.location	否	(none)	String	SSL truststore的位置/文件。
debezium-avro-confluent.ssl.truststore.password	否	(none)	String	SSL truststore的密码。
debezium-avro-confluent.subject	否	(none)	String	用于在序列化期间此格式使用的注册schema的Confluent Schema Registry主题。默认情况下，'kafka'和'upsert-kafka'连接器使用'<topic_name>-value'或'<topic_name>-key'作为默认主题名称，如果此格式用作键或值的格式。但是对于其他连接器（例如'filesystem'），在用作sink时需要使用主题选项。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
debezium-avro-confluent.url	否	(none)	String	用于获取/注册架构的Confluent Schema Registry的URL。

表 2-11 Debezium Json 参数说明

参数	是否必选	默认值	是否必选	描述
format	是	(none)	String	指定要使用的格式，此处应为 'debezium-json'。
debezium-json.schema-include	否	false	Boolean	设置 Debezium Kafka Connect 时，用户可以启用 Kafka 配置 'value.converter.schemas.enable' 以在消息中包含 schema。此选项表明 Debezium JSON 消息是否包含 schema。
debezium-json.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。
debezium-json.timestamp-format.standard	否	'SQL'	String	声明输入和输出的时间戳格式。当前支持的格式为'SQL'和'ISO-8601'。 ● 可选参数 'SQL' 将会以 "yyyy-MM-dd HH:mm:ss.s{precision}" 的格式解析时间戳, 例如 '2020-12-30 12:13:14.123'，且会以相同的格式输出。 ● 可选参数 'ISO-8601' 将会以 "yyyy-MM-ddTHH:mm:ss.s{precision}" 的格式解析输入时间戳, 例如 '2020-12-30T12:13:14.123' ，且会以相同的格式输出。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	是否必选	描述
debezium-json.map-null-key.mode	否	'FAIL'	String	指定处理 Map 中 key 值为空的方法。当前支持的值有FAIL、DROP和LITERAL。 • Option 'FAIL' 将抛出异常，如果遇到 Map 中 key 值为空的数据。 • Option 'DROP' 将丢弃 Map 中 key 值为空的数据项。 • Option 'LITERAL' 将使用字符串常量来替换 Map 中的空 key 值。字符串常量的值由 'debezium-json.map-null-key.literal' 定义。
debezium-json.map-null-key.literal	否	'null'	String	当 'debezium-json.map-null-key.mode' 是 LITERAL 的时候，指定字符串常量替换 Map 中的空 key 值。
debezium-json.encode.decimal-as-plain-number	否	false	Boolean	将所有 DECIMAL 类型的数据保持原状，不使用科学计数法表示。例：0.000000027 默认会表示为 2.7E-8。当此选项设为 true 时，则会表示为 0.000000027。

元数据

元数据可以在 DDL 中作为只读（虚拟）meta 列声明。

Format的元数据只有在相应的连接器使用元数据时才可用。目前，只有Kafka连接器。

表 2-12 元数据

Key	数据类型	说明
schema	STRING NULL	描述payload Schema的JSON字符串。如果Schema不在Debezium记录中，则为NULL。
ingestion-timestamp	TIMESTAMP_LTZ(3) NULL	connector处理事件的时间戳。对应Debezium记录中的ts_ms字段。
source.timestamp	TIMESTAMP_LTZ(3) NULL	源系统创建事件时的时间戳。对应于Debezium记录中的source.ts_ms字段。

版权所有 © 华为技术有限公司

Key	数据类型	说明
source.database	STRING NULL	源数据库。对应于Debezium记录中的source.db字段(如果可用)。
source.schema	STRING NULL	源数据库Schema。对应于Debezium记录中的source.schema字段(如果可用)。
source.table	STRING NULL	源数据库表。对应于Debezium中的source.table或source.collection字段(如果可用)。
source.properties	MAP<STRING, STRING> NULL	各种源属性的Map。对应于Debezium记录中的source字段。

元数据的使用用例参考如下：

```
CREATE TABLE KafkaTable (
  origin_ts TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL,
  event_time TIMESTAMP(3) METADATA FROM 'value.source.timestamp' VIRTUAL,
  origin_database STRING METADATA FROM 'value.source.database' VIRTUAL,
  origin_schema STRING METADATA FROM 'value.source.schema' VIRTUAL,
  origin_table STRING METADATA FROM 'value.source.table' VIRTUAL,
  origin_properties MAP<STRING, STRING> METADATA FROM 'value.source.properties' VIRTUAL,
  user_id BIGINT,
  item_id BIGINT,
  behavior STRING
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'earliest-offset',
  'value.format' = 'debezium-json'
);
```

示例

使用kafka解析Debezium Json数据，并将结果输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink版本为1.15，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
```


版权所有 © 华为技术有限公司

```
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'debezium-json'
);
CREATE TABLE printSink (
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
) WITH (
  'connector' = 'print'
);
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据，该数据表示MySQL 产品表有4列（id、name、description、weight）。该JSON 消息是 products 表上的一条更新事件，其中 id = 111 的行的 weight 值从 5.18 更改为 5.15。

```
{
  "before": {
    "id": 111,
    "name": "scooter",
    "description": "Big 2-wheel scooter",
    "weight": 5.18
  },
  "after": {
    "id": 111,
    "name": "scooter",
    "description": "Big 2-wheel scooter",
    "weight": 5.15
  },
  "source": {
    "version": "0.9.5.Final",
    "connector": "mysql",
    "name": "fullfillment",
    "server_id" :1,
    "ts_sec": 1629607909,
    "gtid": "mysql-bin.000001",
    "pos": 2238,"row": 0,
    "snapshot": false,
    "thread": 7,
    "db": "inventory",
    "table": "test",
    "query": null},
  "op": "u",
  "ts_ms": 1589362330904,
  "transaction": null
}
```

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
-U[111, scooter, Big 2-wheel scooter, 5.18]
+U[111, scooter, Big 2-wheel scooter, 5.15]
```

----结束

 版权所有 © 华为技术有限公司

2.4.7 JSON Format

功能描述

JSON Format 能读写 JSON 格式的数据。当前，JSON schema 是从 table schema 中自动推导而得的。更多具体使用可参考开源社区文档：**JSON Format**。

支持的 Connector

- Kafka
- Upsert Kafka
- Elasticsearch

参数说明

表 2-13

参数	是否必选	默认值	类型	说明
format	是	(none)	String	声明使用的格式，这里应为'json'。
json.fail-on-missing-field	否	false	Boolean	当解析字段缺失时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。
json.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
json.timestamp-format.standard	否	'SQL'	String	声明输入和输出的 TIMESTAMP 和 TIMESTAMP_LTZ 的格式。当前支持的格式为'SQL' 以及 'ISO-8601'： ● 可选参数 'SQL' 将会以 "yyyy-MM-dd HH:mm:ss.s{precision}" 的格式解析 TIMESTAMP, 例如 "2020-12-30 12:13:14.123"，以 "yyyy-MM-dd HH:mm:ss.s{precision}'Z'" 的格式解析 TIMESTAMP_LTZ, 例如 "2020-12-30 12:13:14.123Z" 且会以相同的格式输出。 ● 可选参数 'ISO-8601' 将会以 "yyyy-MM-ddTHH:mm:ss.s{precision}" 的格式解析输入 TIMESTAMP, 例如 "2020-12-30T12:13:14.123" ，以 "yyyy-MM-ddTHH:mm:ss.s{precision}'Z'" 的格式解析 TIMESTAMP_LTZ, 例如 "2020-12-30T12:13:14.123Z" 且会以相同的格式输出。
json.map-null-key.mode	否	'FALL'	String	指定处理 Map 中 key 值为空的方法。当前支持的值有：'FAIL'，'DROP'和'LITERAL'。 ● Option 'FAIL' 将抛出异常，如果遇到 Map 中 key 值为空的数据。 ● Option 'DROP' 将丢弃 Map 中 key 值为空的数据项。 ● Option 'LITERAL' 将使用字符串常量来替换 Map 中的空 key 值。字符串常量的值由 'json.map-null-key.literal' 定义。
json.map-null-key.literal	否	'null'	String	当 'json.map-null-key.mode' 是 LITERAL的时候，指定字符串常量替换 Map 中的空 key 值。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
json.encode.decimal-as-plain-number	否	false	Boolean	将所有 DECIMAL 类型的数据保持原状，不使用科学计数法表示。例：0.000000027 默认会表示为 2.7E-8。当此选项设为 true 时，则会表示为 0.000000027。

数据类型映射

当前，JSON schema 将会自动从 table schema 之中自动推导得到。不支持显式地定义 JSON schema。

在 Flink 中，JSON Format 使用 **jackson databind API** 去解析和生成 JSON。

下表列出了 Flink 中的数据类型与 JSON 中的数据类型的映射关系。

表 2-14 数据类型映射

Flink SQL类型	JSON类型
CHAR / VARCHAR / STRING	string
BOOLEAN	boolean
BINARY / VARBINARY	string with encoding: base64
DECIMAL	number
TINYINT	number
SMALLINT	number
INT	number
BIGINT	number
FLOAT	number
DOUBLE	number
DATE	string with format: date
TIME	string with format: time
TIMESTAMP	string with format: date-time
TIMESTAMP_WITH_LOCAL_TIME_ZONE	string with format: date-time (with UTC time zone)
INTERVAL	number
ARRAY	array
MAP / MULTISET	object

版权所有 © 华为技术有限公司

Flink SQL类型	JSON类型
ROW	object

示例

该示例是从kafka的一个topic中读取数据，并使用kafka sink将数据写入到kafka的另一个topic中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功；否则表示未成功

步骤2 创建flink opensource sql作业，并选择flink版本为1.15，选择保存日志，然后提交并运行，其SQL代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);
insert into printSink select * from kafkaSource;
```

步骤3 向作为source的kafka的topic中插入下列数据：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

 版权所有 © 华为技术有限公司

2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
+I[202103241000000001, webShop, 2021-03-24 10:00:00, 100.0, 100.0, 2021-03-24 10:02:03, 0001, Alice, 330106]
+I[202103241606060001, appShop11, 2021-03-24 16:06:06, 200.0, 180.0, 2021-03-24 16:10:06, 0001, Alice, 330106]
```

----结束

2.4.8 Maxwell Format

功能描述

Maxwell是一个CDC（Changelog Data Capture）工具，可以将MySql中的更改实时流式写入到Kafka等流式connector。Maxwell为changelog提供了统一的格式，而且支持使用JSON对消息进行序列化。

Flink 支持将 Maxwell JSON 消息解释为 INSERT/UPDATE/DELETE 消息到 Flink SQL 系统中。在许多情况下，这对于利用此功能很有用。

例如：

- 将数据库中的增量数据同步到其他系统
- 审计日志
- 数据库的实时物化视图
- 临时连接更改数据库表的历史等等。

Flink 还支持将 Flink SQL 中的 INSERT/UPDATE/DELETE 消息编码为 Maxwell JSON 消息，并发送到 Kafka 等外部系统。但是，目前 Flink 无法将 UPDATE_BEFORE 和 UPDATE_AFTER 合并为一条 UPDATE 消息。因此，Flink 将 UPDATE_BEFORE 和 UDPATE_AFTER 编码为 DELETE 和 INSERT Maxwell 消息。

更多具体使用可参考开源社区文档：**Maxwell Format**。

支持的 Connector

- Kafka
- Filesystem

注意事项

Maxwell应用允许将每个变动消息精确地传递一次。在这种情况下，Flink在消费Maxwell生成的消息时处理得很好。如果Maxwell应用程序在at-least-once模式处理，它可能向Kafka写入重复的改动消息，Flink将获得重复的消息。这可能会导致Flink查询得到错误的结果或意外的异常。因此，在这种情况下，建议将作业配置table.exec.source.cdc-events-duplicate设置为true，并在源表上定义PRIMARY KEY。Framework将生成一个额外的有状态操作符，并使用主键对变更事件进行去重，并生成一个规范化的changelog流。

版权所有 © 华为技术有限公司

参数说明

表 2-15 参数说明

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定使用格式，此处使用'maxwell-json'。
maxwell-json.ignore-parse-errors	否	false	Boolean	跳过解析错误而不是失败的字段和行。出现错误时，字段设置为空。
maxwell-json.timestamp-format.standard	否	'SQL'	String	指定输入和输出时间戳格式。当前支持的值为“SQL”和“ISO-8601”： ● 'SQL'将以“yyyy-MM-dd HH:mm:ss.s{precision}”格式解析输入时间戳，例如'2020-12-30 12:13:14.123' 并以相同格式输出时间戳。 ● 'ISO-8601'将以“yyyy-MM-ddTHH:mm:ss.s{precision}”格式解析输入时间戳，例如'2020-12-30T12:13:14.123' 并以相同格式输出时间戳。
maxwell-json.map-null-key.mode	否	'FAIL'	String	指定序列化map数据的null键时的处理模式。当前支持的值为“FAIL”、“DROP”和“LITERAL”： ● 'FAIL'将在遇到带有null键的map时抛出异常。 ● 'DROP'将删除map数据的null键条目。 ● 'LITERAL'将使用字符串代替null键。字符串由 maxwell-json.map-null-key.literal 选项定义。
maxwell-json.map-null-key.literal	否	'null'	String	当 'maxwell-json.map-null-key.mode' 为 LITERAL 时，指定字符串以替换null键。
maxwell-json.encode.decimal-as-plain-number	否	false	Boolean	将所有小数编码为普通数字，而不是可能的科学计数法。默认情况下，小数可以使用科学计数法书写。例如，0.000000027在默认情况下被编码为2.7E-8，如果将此选项设置为true，则将被写入为0.000000027。

版权所有 © 华为技术有限公司

元数据

元数据可以在 DDL 中作为只读（虚拟）meta 列声明。

表 2-16 元数据

Key	数据类型	说明
database	STRING NULL	源数据库。对应于Maxwell记录中的数据库字段（如果可用）。
table	STRING NULL	源数据库中的表。对应于Maxwell记录中的表字段（如果可用）。
primary-key-columns	ARRAY<STRING> NULL	主键名数组。对应于Maxwell记录中的primary_key_columns字段（如果可用）。
ingestion-timestamp	TIMESTAMP_LTZ(3) NULL	连接器处理事件的时间戳。对应Maxwell记录中的ts字段。

元数据使用示例如下：

```
CREATE TABLE KafkaTable (
  origin_database STRING METADATA FROM 'value.database' VIRTUAL,
  origin_table STRING METADATA FROM 'value.table' VIRTUAL,
  origin_primary_key_columns ARRAY<STRING> METADATA FROM 'value.primary-key-columns' VIRTUAL,
  origin_ts TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL,
  user_id BIGINT,
  item_id BIGINT,
  behavior STRING
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'earliest-offset',
  'value.format' = 'maxwell-json'
);
```

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.15，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  id bigint,
  name string,
  description string,
```


版权所有 © 华为技术有限公司

```
  weight DECIMAL(10, 2)
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'maxwell-json'
);


CREATE TABLE printSink (
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
) WITH (
  'connector' = 'print'
);
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据（每个字段的含义请参考**Maxwell documentation**）：

```
{
  "database":"test",
  "table":"e",
  "type":"insert",
  "ts":1477053217,
  "xid":23396,
  "commit":true,
  "position":"master.000006:800911",
  "server_id":23042,
  "thread_id":108,
  "primary_key": [1, "2016-10-21 05:33:37.523000"],
  "primary_key_columns": ["id", "c"],
  "data":{
    "id":111,
    "name":"scooter",
    "description":"Big 2-wheel scooter",
    "weight":5.15
  },
  "old":{
    "weight":5.18
  }
}
```

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
+I[111, scooter, Big 2-wheel scooter, 5.15]
```

----结束

2.4.9 Ogg Format

功能描述

Oracle GoldenGate (a.k.a ogg) 是一个实现异构 IT 环境间数据实时数据集成和复制的综合软件包。 该产品集支持高可用性解决方案、实时数据集成、事务更改数据捕

获、运营和分析企业系统之间的数据复制、转换和验证。Ogg 为变更日志提供了统一的格式结构，并支持使用 JSON 序列化消息。

Flink 支持将 Ogg JSON 消息解析为 INSERT/UPDATE/DELETE 消息到 Flink SQL 系统中。在很多情况下，利用这个特性非常有用，例如

- 将增量数据从数据库同步到其他系统
- 日志审计
- 数据库的实时物化视图
- 关联维度数据库的变更历史，等等

Flink 还支持将 Flink SQL 中的 INSERT/UPDATE/DELETE 消息编码为 Ogg JSON 格式的消息, 输出到 Kafka 等存储中。 但需要注意, 目前 Flink 还不支持将 UPDATE_BEFORE 和 UPDATE_AFTER 合并为一条 UPDATE 消息. 因此, Flink 将 UPDATE_BEFORE 和 UPDATE_AFTER 分别编码为 DELETE 和 INSERT 类型的 Ogg 消息。

支持的 Connector

- Kafka
- FileSystem

参数说明

表 2-17 参数说明

参数	是否必须	默认值	类型	描述
format	是	(none)	String	指定要使用的格式，此处应为 'ogg-json'。
ogg-json.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。
debezium-json.timestamp-format.standard	否	'SQL'	String	声明输入和输出的时间戳格式。当前支持的格式为'SQL' 以及 'ISO-8601'： ● 可选参数 'SQL' 将会以 "yyyy-MM-dd HH:mm:ss.s{precision}" 的格式解析时间戳, 例如 '2020-12-30 12:13:14.123'，且会以相同的格式输出。 ● 可选参数 'ISO-8601' 将会以 "yyyy-MM-ddTHH:mm:ss.s{precision}" 的格式解析输入时间戳, 例如 '2020-12-30T12:13:14.123' ，且会以相同的格式输出。

版权所有 © 华为技术有限公司

参数	是否必须	默认值	类型	描述
ogg-json.map-null-key.mode	否	'FAIL'	String	指定处理 Map 中 key 值为空的方法. 当前支持的值有 'FAIL', 'DROP' 和 'LITERAL': ● Option 'FAIL' 将抛出异常。 ● Option 'DROP' 将丢弃 Map 中 key 值为空的数据项。 ● Option 'LITERAL' 将使用字符串常量来替换 Map 中的空 key 值。字符串常量的值由 ogg-json.map-null-key.literal 定义。
ogg-json.map-null-key.literal	否	'null'	String	当 'ogg-json.map-null-key.mode' 是 LITERAL 的时候，指定字符串常量替换 Map 中的空 key 值。

元数据

表 2-18 元数据

Key	数据类型	描述
table	STRING NULL	包含完全限定的表名。完全限定表名的格式为：CATALOG NAME.SCHEMA NAME.TABLE NAME
primary-keys	ARRAY<STRING> NULL	保存源表的主键的列名的数组。 如果includePrimaryKeys配置属性设置为true，则仅在JSON输出中包含primary-keys字段。
ingestion-timestamp	TIMESTAMP_LTZ(6) NULL	connector处理事件的时间戳。对应Ogg记录中的current_ts字段。
event-timestamp	TIMESTAMP_LTZ(6) NULL	源系统创建事件的时间戳。对应Ogg记录的op_ts字段。

元数据的使用用例参考如下：

```
CREATE TABLE KafkaTable (
  origin_ts TIMESTAMP(3) METADATA FROM 'value.ingestion-timestamp' VIRTUAL,
  event_time TIMESTAMP(3) METADATA FROM 'value.event-timestamp' VIRTUAL,
  origin_table STRING METADATA FROM 'value.table' VIRTUAL,
  primary_keys ARRAY<STRING> METADATA FROM 'value.primary_keys' VIRTUAL,
  user_id BIGINT,
  item_id BIGINT,
  behavior STRING
) WITH (
  'connector' = 'kafka',
```


版权所有 © 华为技术有限公司

```
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'earliest-offset',
  'value.format' = 'ogg-json'
);
```

示例

使用ogg-json读取kafka中的ogg记录，并输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.15版本，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'ogg-json'
);

CREATE TABLE printSink (
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
) WITH (
  'connector' = 'print'
);
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据，该数据表示Oracle PRODUCTS 表 有 4 列 (id, name, description and weight). 上面的 JSON 消息是 PRODUCTS 表上的一条更新事件，其中 id = 111 的行的 weight 值从 5.18 更改为 5.15。

```
{
  "before": {
    "id": 111,
    "name": "scooter",
    "description": "Big 2-wheel scooter",
    "weight": 5.18
  },
  "after": {
    "id": 111,
    "name": "scooter",
    "description": "Big 2-wheel scooter",
    "weight": 5.15
  },
  "op_type": "U",
  "op_ts": "2020-05-13 15:40:06.000000",
  "current_ts": "2020-05-13 15:40:07.000000",
  "primary_keys": [
    "id"
  ],
  "pos": "00000000000000000000143",
  "table": "PRODUCTS"
}
```

 版权所有 © 华为技术有限公司

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
-U[111, scooter, Big 2-wheel scooter, 5.18]
+U[111, scooter, Big 2-wheel scooter, 5.15]
```

----结束

2.4.10 Orc Format

功能描述

Apache Orc Format允许读写ORC数据。更多具体使用可参考开源社区文档：**Orc Format**。

支持的 Connector

- FileSystem

参数说明

表 2-19 参数说明

参数	是否必选	默认值	类型	描述
format	是	无	String	指定要使用的格式，这里应该是 'orc'。

Orc 格式也支持来源于 **Table properties** 的表属性。 举个例子，您可以设置 orc.compress=SNAPPY 来允许spappy压缩。

数据类型映射

Orc 格式类型的映射和 Apache Hive 是兼容的。下面的表格列出了 Flink 类型的数据和 Orc 类型的数据的映射关系。

表 2-20 数据类型映射

Flink数据类型	Orc物理类型	Orc逻辑类型
CHAR	bytes	CHAR
VARCHAR	bytes	VARCHAR
STRING	bytes	STRING
BOOLEAN	long	BOOLEAN

版权所有 © 华为技术有限公司

Flink数据类型	Orc物理类型	Orc逻辑类型
BYTES	bytes	BINARY
DECIMAL	decimal	DECIMAL
TINYINT	long	BYTE
SMALLINT	long	SHORT
INT	long	INT
BIGINT	long	LONG
FLOAT	double	FLOAT
DOUBLE	double	DOUBLE
DATE	long	DATE
TIMESTAMP	timestamp	TIMESTAMP
ARRAY	-	LIST
MAP	-	MAP
ROW	-	STRUCT

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，**开启checkpoint**，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic-pattern' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv'
);


CREATE TABLE sink (
  order_id string,
  order_channel string,
```


版权所有 © 华为技术有限公司

```
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'filesystem',
  'format' = 'orc',
  'path' = 'obs://xx'
);
insert into sink select * from kafkaSource;
```

步骤3 向kafka的作为source的topic中插入下列数据：

```
20210325150505001,appshop,2021-03-25 15:05:05,500.00,400.00,2021-03-25 15:10:00,0003,Cindy,330108

20210324160606001,appShop,2021-03-24 16:06:06,200.00,180.00,2021-03-24 16:10:06,0001,Alice,330106
```

步骤4 读取sink表中配置的obs路径中的orc文件，其数据结果如下

```
20210325120202001, miniAppShop, 2021-03-25 12:02:02, 60.0, 60.0, 2021-03-25 12:03:00, 0002, Bob, 330110

20210324160606001, appShop, 2021-03-24 16:06:06, 200.0, 180.0, 2021-03-24 16:10:06, 0001, Alice, 330106
```

----结束

2.4.11 Parquet Format

功能描述

Apache Parquet格式允许读写 Parquet 数据。更多具体使用可参考开源社区文档：**Parquet Format**。

支持的 Connector

- FileSystem

参数说明

表 2-21 参数说明

参数	是否必选	默认值	类型	描述
format	是	无	String	指定使用的格式，此处应为 "parquet"。
parquet.utc-timezone	否	false	Boolean	使用 UTC 时区或本地时区在纪元时间和 LocalDateTime 之间进行转换。Hive 0.x/1.x/2.x 使用本地时区，但 Hive 3.x 使用 UTC 时区。

数据类型映射

目前，Parquet 格式类型映射与 Apache Hive 兼容，但与 Apache Spark 有所不同：

 版权所有 © 华为技术有限公司

- Timestamp：不论精度，映射 timestamp 类型至 int96。
- Decimal：根据精度，映射 decimal 类型至固定长度字节的数组。

下表列举了 Flink 中的数据类型与 JSON 中的数据类型的映射关系。

注意：复合数据类型暂只支持写不支持读（Array、Map 与 Row）。

表 2-22 数据类型映射

Flink数据类型	Parquet类型	Parquet逻辑类型
CHAR / VARCHAR / STRING	BINARY	UTF8
BOOLEAN	BOOLEAN	-
BINARY / VARBINARY	BINARY	-
DECIMAL	FIXED_LEN_BYTE_ARRAY	DECIMAL
TINYINT	INT32	INT_8
SMALLINT	INT32	INT_16
INT	INT32	-
BIGINT	INT64	-
FLOAT	FLOAT	-
DOUBLE	DOUBLE	-
DATE	INT32	DATE
TIME	INT32	TIME_MILLIS
TIMESTAMP	INT96	-
ARRAY	-	LIST
MAP	-	MAP
ROW	-	STRUCT

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列> 找到作业的所属队列> 更多> 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，**开启checkpoint**，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
```

 版权所有 © 华为技术有限公司

```
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic-pattern' = 'kafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);


CREATE TABLE sink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'filesystem',
  'format' = 'parquet',
  'path' = 'obs://xx'
);
insert into sink select * from kafkaSource;
```

步骤3 向kafka的作为source的topic中插入下列数据：

```
20210325150505001,appShop,2021-03-25 15:05:05,500.00,400.00,2021-03-25 15:10:00,0003,Cindy,330108

20210324160606001,appShop,2021-03-24 16:06:06,200.00,180.00,2021-03-24 16:10:06,0001,Alice,330106
```

步骤4 读取sink表中配置的obs路径中的parquet文件，其数据结果如下

```
20210325120202001, miniAppShop, 2021-03-25 12:02:02, 60.0, 60.0, 2021-03-25 12:03:00, 0002, Bob, 330110

20210324160606001, appShop, 2021-03-24 16:06:06, 200.0, 180.0, 2021-03-24 16:10:06, 0001, Alice, 330106
```

----结束

2.4.12 Raw Format

功能描述

Raw format 允许读写原始（基于字节）值作为单个列。

说明

- Raw Format将 null 值编码成 byte[] 类型的 null。这样在 upsert-kafka 中使用时可能会有限制，因为 upsert-kafka 将 null 值视为 墓碑消息（在键上删除）。因此，如果该字段可能具有 null 值，我们建议避免使用 upsert-kafka 连接器和 raw format 作为 value.format。
- Raw format 连接器是内置的。更多具体使用可参考开源社区文档：**Raw Format**。

支持的 Connector

- Kafka

- Upsert Kafka
- Filesystem

参数说明

表 2-23

参数	是否必选	默认值	类型	描述
format	是	(none)	String	指定要使用的格式, 这里应该是'raw'。
raw.charset	否	UTF-8	String	指定字符集来编码文本字符串。
raw.endianness	否	big-endian	String	指定字节序来编码数字值的字节。有效值为'big-endian'和'little-endian'。

数据类型映射

下表详细说明了这种格式支持的 SQL 类型，包括用于编码和解码的序列化类和反序列化类的详细信息。

表 2-24 数据类型映射

Flink SQL 类型	值
CHAR / VARCHAR / STRING	UTF-8（默认）编码的文本字符串。编码字符集可以通过 'raw.charset' 进行配置。
BINARY / VARBINARY / BYTES	字节序列本身。
BOOLEAN	表示布尔值的单个字节，0表示 false, 1表示 true。
TINYINT	有符号数字值的单个字节。
SMALLINT	采用big-endian（默认）编码的两个字节。字节序可以通过 'raw.endianness' 配置。
INT	采用 big-endian （默认）编码的四个字节。字节序可以通过 'raw.endianness' 配置。
BIGINT	采用 big-endian （默认）编码的八个字节。字节序可以通过 'raw.endianness' 配置。

版权所有 © 华为技术有限公司

Flink SQL 类型	值
FLOAT	采用 IEEE 754 格式和 big-endian （默认）编码的四个字节。字节序可以通过 'raw.endianness' 配置。
DOUBLE	采用 IEEE 754 格式和 big-endian （默认）编码的八个字节。字节序可以通过 'raw.endianness' 配置。
RAW	通过 RAW 类型的底层 TypeSerializer 序列化的字节序列。

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列 > 找到作业的所属队列 > 更多 > 测试地址连通性 > 输入kafka的地址 > 测试）。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.15，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  log string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'raw'
);

CREATE TABLE printSink (
  log string
) WITH (
  'connector' = 'print'
);
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据：

```
47.29.201.179 - - [28/Feb/2019:13:17:10 +0000] "GET /?p=1 HTTP/2.0" 200 5316 "https://domain.com/?p=1" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36" "2.75"
```

步骤4 按照如下方式查看taskmanager.out文件中的数据结果：

1. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
2. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
3. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取.out文件查看结果日志。

```
+I[47.29.201.179 - - [28/Feb/2019:13:17:10 +0000] "GET /?p=1 HTTP/2.0" 200 5316 "https://domain.com/?p=1" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36" "2.75"]
```

----结束

 版权所有 © 华为技术有限公司

2.5 Connector 列表

2.5.1 Connector 概述

表类型

- 源表：源表是Flink作业的数据输入表，例如Kafka等实时流数据输入。
- 维表：数据源表的辅助表，用于丰富和扩展源表的数据。在Flink作业中，因为数据采集端采集到的数据往往比较有限，在做数据分析之前，就要先将所需的维度信息补全，而维表就是代表存储数据维度信息的数据源。常见的用户维表有MySQL，Redis等。
- 结果表：Flink作业输出的结果数据表，将每条实时处理完的数据写入的目标存储，如 MySQL，HBase 等数据库。

示例：

Flink 实时消费用户订单数据的 Kafka 源表，通过Redis维表将商品ID关联维表获取商品分类，并计算不同类别的商品销售金额，将计算结果写入 RDS（Relational Database Service，如 MySQL）结果表中。

表信息如下：

- 源表：订单数据表，包含用户ID、商品ID、订单ID、订单金额等信息。
- 维表：用户信息表，包含商品ID、商品类别信息。
- 结果表：按商品类别统计订单销售金额数据。

作业首先从订单数据源表读取实时订单数据，将订单数据流与商品商品类别信息维表关联起来，然后聚合统计订单总额，最后将统计结果写入结果表。

本例中订单表作为驱动源表输入，商品类别信息表作为静态维表，统计结果表作为作业最终输出。

Connector 支持列表

表 2-25 Connector 支持列表

Connector	源表	维表	结果表
BlackHole	不支持	不支持	支持
ClickHouse	不支持	不支持	支持
DataGen	支持	不支持	不支持
Doris	支持	支持	支持
DWS	支持	支持	支持
Elasticsearch	不支持	不支持	支持
FileSystem	支持	不支持	支持

版权所有 © 华为技术有限公司

Connector	源表	维表	结果表
Hbase	支持	支持	支持
Hive	支持	支持	支持
JDBC	支持	支持	支持
Kafka	支持	不支持	支持
Print	不支持	不支持	支持
Redis	支持	支持	支持
Upsert Kafka	支持	不支持	支持

2.5.2 BlackHole

功能描述

BlackHole Connector允许接收所有输入记录，常用于高性能测试和UDF输出，其不是实质性Sink。Blackhole结果表是系统内置的Connector。

例如，如果您在注册其他类型的Connector结果表时报错，但您不确定是系统问题还是结果表WITH参数错误，您可以将WITH参数修改为'connector' = 'blackhole'后，单击运行。如果不再报错，则证明系统没有问题，您需要排查确认修改WITH参数是否正确。

表 2-26 支持类别

类别	详情
支持表类型	结果表

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**

语法格式

```
create table blackhole_table (
 attr_name attr_type (',' attr_name attr_type) *
) with (
 'connector' = 'blackhole'
);
```


版权所有 © 华为技术有限公司

参数说明

表 2-27 参数说明

选项	是否必要	默认值	类型	描述
connector	是	无	String	指定需要使用的连接器，此处应为'blackhole'。

示例

通过DataGen源表产生数据，BlackHole结果表接收传来的数据。

```
create table datagenSource (
 user_id string,
 user_name string,
 user_age int
) with (
 'connector' = 'datagen',
 'rows-per-second'='1'
);
create table blackholeSink (
 user_id string,
 user_name string,
 user_age int
) with (
 'connector' = 'blackhole'
);
insert into blackholeSink select * from datagenSource;
```

2.5.3 ClickHouse

功能描述

DLI支持将Flink作业数据输出到ClickHouse数据库中，表类型仅支持结果表。

ClickHouse是面向联机分析处理的列式数据库，支持SQL查询，且查询性能好，特别是基于大宽表的聚合分析查询性能非常优异，比其他分析型数据库速度快一个数量级。详细请参考**ClickHouse组件操作**。

表 2-28 支持类别

类别	详情
支持表类型	结果表

前提条件

- 该场景作业需要运行在DLI的独享队列上。
- 该场景需要与ClickHouse建立增强型跨源连接，并根据实际情况设置ClickHouse集群所在安全组规则中的端口。

  如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。

版权所有 © 华为技术有限公司

如何设置安全组规则，请参见《虚拟私有云用户指南》中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 创建MRS的ClickHouse集群，集群版本选择MRS 3.1.0及以上版本。
- ClickHouse结果表不支持删除表数据操作。
- Flink中支持字段类型范围为：string、tinyint、smallint、int、bigint、float、double、date、timestamp、decimal以及Array。

  其中Array中的数据类型仅支持int、bigint、string、float、double。

语法格式

```
create table clickhouseSink (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'type' = 'clickhouse',
  'url' = '',
  'table-name' = ''
);
```

参数说明

表 2-29 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	固定为：clickhouse

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
url	是	无	String	ClickHouse的url。 参数格式为：jdbc:clickhouse://**ClickHouseBalancer实例业务IP1**:**ClickHouseBalancer端口**,**ClickHouseBalancer实例业务IP2**:**ClickHouseBalancer端口/数据库名** • ClickHouseBalancer实例的IP地址： 登录MRS管理控制台，选择“集群名称 > 组件管理 > ClickHouse > 实例”，获取ClickHouseBalancer实例的业务IP。 • ClickHouseBalancer端口： 登录MRS管理控制台，选择“集群名称 > 组件管理 > ClickHouse > 服务配置”，角色选择“ClickHouseBalancer”。当MRS集群未开启Kerberos认证时，搜索“lb_http_port”配置参数值，默认值为21425；当开启Kerberos认证时，搜索“lb_https_port”配置参数值，默认值为21426。 • 数据库名为ClickHouse集群创建的数据库名称。如果数据库名不存在，则不需要填写。 • 建议配置多个ClickHouseBalancer实例IP以避免ClickHouseBalancer实例单点故障。 • MRS集群开启开启Kerberos认证时，还需要在url中加上ssl、sslmode请求参数，将ssl设为true，sslmode设为none，示例见**示例2**。
table-name	是	无	String	ClickHouse的表名。
driver	否	ru.yandex.clickhouse.ClickHouseDriver	String	连接数据库所需要的驱动。若未配置，则会自动通过URL提取，默认为ru.yandex.clickhouse.ClickHouseDriver。
username	否	无	String	访问ClickHouse数据库的账号名，MRS集群开启Kerberos认证时需要填写。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
password	否	无	String	访问ClickHouse数据库账号的密码，MRS集群开启Kerberos认证时需要填写。
sink.buffer-flush.max-rows	否	100	Integer	写数据时刷新数据的最大行数，默认值为100。
sink.buffer-flush.interval	否	1s	Duration	刷新数据的时间间隔，单位可以为ms、milli、millisecond/s、sec、second/min、minute等，默认值为1s。设置为0则表示不根据时间刷新。
sink.max-retries	否	3	Integer	写数据失败时的最大尝试次数，默认值为3。

示例

- **示例1：从Kafka中读取数据，并将数据插入ClickHouse中（ClickHouse版本为MRS的21.3.4.25，且MRS集群未开启Kerberos认证）：**
  a. 参考**增强型跨源连接**，在DLI上根据ClickHouse和Kafka集群所在的虚拟私有云和子网分别创建跨源连接，并绑定所要使用的Flink弹性资源池。
  b. 设置ClickHouse和Kafka集群安全组的入向规则，使其对当前将要使用的Flink作业队列网段放通。参考**测试地址连通性**根据ClickHouse和Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
  c. 使用ClickHouse客户端连接到ClickHouse服务端，并使用以下命令查询集群标识符cluster等其他环境参数信息。
  详细操作请参考**从零开始使用ClickHouse**。

```
select cluster,shard_num,replica_num,host_name from system.clusters;
```

  其返回信息如下图：

```
┌─cluster─────────┬─shard_num─┐
│ default_cluster │         1 │
│ default_cluster │         2 │
└─────────────────┴───────────┘
```

  根据获取到的集群标识符cluster，例如当前为default_cluster ，使用以下命令在ClickHouse的default_cluster集群节点上创建数据库flink。

```
CREATE DATABASE flink ON CLUSTER default_cluster;
```

  d. 使用以下命令在default_cluster集群节点上和flink数据库下创建表名为order的ReplicatedMergeTree表。

```
CREATE TABLE flink.order ON CLUSTER default_cluster(order_id String,order_channel String,order_time String,pay_amount Float64,real_pay Float64,pay_time String,user_id String,user_name String,area_id String) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/flink/order', '{replica}')ORDER BY order_id;
```

  e. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将DMS Kafka作为数据源，ClickHouse作业结果表。
  如下脚本中的加粗参数请根据实际环境修改。

版权所有 © 华为技术有限公司

```
create table orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

create table clickhouseSink(
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) with (
  'connector' = 'clickhouse',
  'url' = 'jdbc:clickhouse://
***ClickhouseAddress1:ClickhousePort,ClickhouseAddress2:ClickhousePort***/flink',
  'username' = '**username**',
  'password' = '**password**',
  'table-name' = 'order',
  'sink.buffer-flush.max-rows' = '10',
  'sink.buffer-flush.interval' = '3s'
);

insert into clickhouseSink select * from orders;
```

f. 连接Kafka集群，向DMS Kafka中插入以下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

g. 使用ClickHouse客户端连接到ClickHouse，执行以下查询命令，查询写入flink数据库下order表中的数据。

```
select * from flink.order;
```

查询结果参考如下：

```
202103241000000001 webShop 2021-03-24 10:00:00 100 100 2021-03-24 10:02:03 0001 Alice 330106

202103241606060001 appShop 2021-03-24 16:06:06 200 180 2021-03-24 16:10:06 0001 Alice 330106

202103251202020001 miniAppShop 2021-03-25 12:02:02 60 60 2021-03-25 12:03:00 0002 Bob 330110
```

 版权所有 © 华为技术有限公司

- **示例2：从Kafka中读取数据，并将数据插入ClickHouse中（ClickHouse版本为MRS的21.3.4.25，且MRS集群开启Kerberos认证）**

  a. 参考**增强型跨源连接**，在DLI上根据ClickHouse和Kafka集群所在的虚拟私有云和子网分别创建跨源连接，并绑定所要使用的Flink弹性资源池。

  b. 设置ClickHouse和Kafka集群安全组的入向规则，使其对当前将要使用的Flink作业队列网段放通。参考**测试地址连通性**根据ClickHouse和Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。

  c. 使用ClickHouse客户端连接到ClickHouse服务端，并使用以下命令查询集群标识符cluster等其他环境参数信息。

  参考**从零开始使用ClickHouse**。

  ```
  select cluster,shard_num,replica_num,host_name from system.clusters;
  ```

  其返回信息如下图：

  ```
  ┌─cluster─────────┬─shard_num─┐
  │ default_cluster │         1 │
  │ default_cluster │         2 │
  └─────────────────┴───────────┘
  ```

  根据获取到的集群标识符cluster，例如当前为default_cluster ，使用以下命令在ClickHouse的default_cluster集群节点上创建数据库flink。

  ```
  CREATE DATABASE flink ON CLUSTER default_cluster;
  ```

  d. 使用以下命令在default_cluster集群节点上和flink数据库下创建表名为order的ReplicatedMergeTree表。

  ```
  CREATE TABLE flink.order ON CLUSTER default_cluster(order_id String,order_channel String,order_time String,pay_amount Float64,real_pay Float64,pay_time String,user_id String,user_name String,area_id String) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/flink/order', '{replica}')ORDER BY order_id;
  ```

  e. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将Kafka作为数据源，ClickHouse作业结果表。

  如下脚本中的加粗参数请根据实际环境修改。

  ```
  create table orders (
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string
  ) WITH (
    'connector' = 'kafka',
    'topic' = '***KafkaTopic***',
    'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
    'properties.group.id' = '***GroupId***',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'json'
  );

  create table clickhouseSink(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string
  ```


版权所有 © 华为技术有限公司

```
) with (
  'connector' = 'clickhouse',
  'url' = 'jdbc:clickhouse://
ClickhouseAddress1:ClickhousePort,ClickhouseAddress2:ClickhousePort/flink?
ssl=true&sslmode=none',
  'table-name' = 'order',
  'username' = 'username',
  'password' = 'password', --DEW凭据中的key
  'sink.buffer-flush.max-rows' = '10',
  'sink.buffer-flush.interval' = '3s',
  'dew.endpoint'='kms.xx.xx.com', --使用的DEW服务所在的endpoint信息
  'dew.csms.secretName'='xx', --DEW服务通用凭据的凭据名称
  'dew.csms.decrypt.fields'='password', --password字段值需要利用DEW凭证管理,进行解密替换
  'dew.csms.version'='v1'
);

insert into clickhouseSink select * from orders;
```

f. 连接Kafka集群，向Kafka中插入以下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24
10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03",
"user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24
16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06",
"user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

g. 使用ClickHouse客户端连接到ClickHouse，执行以下查询命令，查询写入flink数据库下order表中的数据。

```
select * from flink.order;
```

查询结果参考如下：

```
202103241000000001 webShop 2021-03-24 10:00:00 100 100 2021-03-24 10:02:03 0001 Alice
330106

202103241606060001 appShop 2021-03-24 16:06:06 200 180 2021-03-24 16:10:06 0001 Alice
330106

202103251202020001 miniAppShop 2021-03-25 12:02:02 60 60 2021-03-25 12:03:00 0002 Bob
330110
```

2.5.4 DataGen

功能描述

DataGen主要用于生成随机数据，可用于调试以及测试等场景。

表 2-30 支持类别

类别	详情
支持表类型	源表

注意事项

- 创建DataGen表时，表字段类型不支持Array，Map和Row复杂类型，可以通过**CREATE TABLE语句**中的“**COMPUTED COLUMN**”来进行类似功能构造。

版权所有 © 华为技术有限公司

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**

语法格式

```
create table dataGenSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)
)
with (
  'connector' = 'datagen'
);
```

参数说明

表 2-31 参数说明

参数	是否必选	默认值	数据类型	参数说明
connector	是	无	String	指定要使用的连接器，这里是'datagen'。
rows-per-second	否	10000	Long	每秒生成的行数，用以控制数据发出速率。
number-of-rows	否	无	Long	生成数据的总行数。默认条件下，不限制生成数据的总行数。如果有字段生成器类型为序列生成器，则当生成数据的行数达到上限或者序列数字达到结束值时，都不会再生成数据。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
fields.#.kind	否	random	String	指定 '#' 字段的生成器。 **'#' 字段必须是DataGen表中的字段，实际使用时需要将'#'替换为相应字段名。**其他各参数的'#'号意义相同，不再重复描述。 参数值可以是 'sequence' 或 'random'，具体含义如下： ● random是默认值，表示无界的随机生成器。您可以通过“fields.#.max”和“fields.#.min”参数指定随机生成数的最大和最小值。当指定的字段类型为char、varchar、string时，可以通过“fields.#.length”参数指定长度。当指定的字段类型为时间戳类型时，可以通过“fields.#.max-past”参数指定相对当前时间向过去偏移的最大值。 ● sequence表示有界的序列生成器。您可以通过“fields.#.start”和“fields.#.end”指定序列的起始和结束值，当序列数字达到结束值时，就不会再生成数据。
fields.#.min	否	'#'号指定的字段类型的最小值	'#'号指定的字段类型	当“fields.#.kind”字段为：random时有效。 表示随机生成器的最小值，'#' 指定的字段仅适用于数字类型。
fields.#.max	否	'#'号指定的字段类型的最大值	'#'号指定的字段类型	当“fields.#.kind”字段为：random时有效。 随机生成数的最大值，'#' 指定的字段仅适用于数字类型。
fields.#.max-past	否	0	Duration	当“fields.#.kind”字段为：random时有效。 随机生成器生成相对当前时间向过去偏移的最大值，'#' 指定的字段仅适用于时间戳类型。
fields.#.length	否	100	Integer	当“fields.#.kind”字段为：random时有效。 随机生成器生成字符的长度，'#' 指定的字段仅适用于char、varchar、string。
fields.#.start	否	无	'#'号指定的字段类型	当“fields.#.kind”字段为：sequence时有效。 序列生成器的起始值。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
fields.#.end	否	无	'#'号指定的字段类型	当“fields.#.kind”字段为：sequence时有效。 序列生成器的结束值。

示例

参考**创建Flink OpenSource作业**，创建flink opensource sql作业，运行如下作业脚本，通过DataGen表产生随机数据并输出到Print结果表中。

```
create table dataGenSource(
  user_id string,
  amount int
) with (
  'connector' = 'datagen',
  'rows-per-second' = '1', --每秒生成一条数据
  'fields.user_id.kind' = 'random', --为字段user_id指定random生成器
  'fields.user_id.length' = '3', --限制字段user_id长度为3
  'fields.amount.kind' = 'sequence', --为字段amount指定sequence生成器
  'fields.amount.start' = '1', --字段amount的起始值
  'fields.amount.end' = '1000' --字段amount的结束值
);

create table printSink(
  user_id string,
  amount int
) with (
  'connector' = 'print'
);

insert into printSink select * from dataGenSource;
```

该作业提交后，作业状态变成“运行中”，后续您可通过如下操作查看输出结果。

- 方法一：
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
  c. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
- 方法二：如果在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
  c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载文件名包含taskmanager.out的文件查看结果日志。

2.5.5 Doris

版权所有 © 华为技术有限公司

2.5.5.1 Doris Connector 概述

Flink Doris Connector 可以支持通过 Flink 操作（读取、插入、修改、删除） Doris 中存储的数据。详情参考**Flink Doris Connector**。

说明

只能对Unique Key模型的表进行修改和删除操作。

表 2-32 支持类别

类别	详情
支持表类型	源表、维表、结果表

2.5.5.2 Doris 源表

功能描述

Flink SQL作业读取Doris源表。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与Doris建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- **如果使用MRS Doris，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《数据湖探索用户指南》中的“**修改主机信息**”章节描述。
- 集群未启用Kerberos认证（普通模式）

  使用**admin**用户连接Doris后，创建具有管理员权限的角色并绑定给用户。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 集群未启用Kerberos认证（普通模式）
- Doris的表名是区分大小写。
- 使用CloudTable的doris时，'fenodes'字段值的端口请用8030，如'xx:8030'。同时安全组请放开端口8030, 8040，9030。
- 开启HTTPS后，需要在创建表的**with**子句中添加如下配置参数：

版权所有 © 华为技术有限公司

- **'doris.enable.https' = 'true'**
- **'doris.ignore.https.ca' = 'true'**

语法格式

```
create table dorisSource (
  attr_name attr_type
  (',' attr_name attr_type)*
 )
with (
  'connector' = 'doris',
  'fenodes' = 'FE_IP:PORT,FE_IP:PORT,FE_IP:PORT',
  'table.identifier' = 'database.table',
  'username' = 'dorisUsername',
  'password' = 'dorisPassword'
);
```

参数说明

通用配置项

参数	默认值	是否必选	参数类型说明
fenodes	--	是	Doris FE ip地址和port, 多实例之间使用逗号分隔。其中port可登录MRS Manager，选择“集群 > 服务 > Doris > 配置”，在搜索框中搜索“http”查看。如果开启https，则搜索“https”。
table.identifier	--	是	Doris表名，如：db.tbl
username	--	是	访问Doris的用户名。
password	--	是	访问Doris的密码。
doris.request.retries	3	否	向Doris发送请求的重试次数。
doris.request.connect.timeout.ms	30000	否	向Doris发送请求的连接超时时间。
doris.request.read.timeout.ms	30000	否	向Doris发送请求的读取超时时间。
doris.request.query.timeout.s	3600	否	查询Doris的超时时间，默认值为1小时，-1表示无超时限制。
doris.request.tablet.size	Integer.MAX_VALUE	否	一个 Partition 对应的Doris Tablet 个数。 此数值设置越小，则会生成越多的 Partition。从而提升 Flink 侧的并行度，但同时会对 Doris 造成更大的压力。
doris.batch.size	1024	否	一次从 BE 读取数据的最大行数。增大此数值可减少Flink与Doris之间建立连接的次数。 从而减轻网络延迟所带来的额外时间开销。

版权所有 © 华为技术有限公司

参数	默认值	是否必选	参数类型说明
doris.exec.mem.limit	2147483648	否	单个查询的内存限制。默认为 2GB，单位为字节。
doris.deserialize.arrow.async	FALSE	否	是否支持异步转换 Arrow 格式到 flink-doris-connector 迭代所需的 RowBatch。
doris.deserialize.queue.size	64	否	异步转换 Arrow 格式的内部处理队列，当doris.deserialize.arrow.async 为 true 时生效。
doris.read.field	--	否	读取 Doris 表的列名列表，多列之间使用逗号分隔。
doris.filter.query	--	否	过滤读取数据的表达式，此表达式透传给 Doris。Doris 使用此表达式完成源端数据过滤。

示例

该示例是从Doris源表读取数据，并输入到 print connector。

1. 参考**增强型跨源连接**，在DLI上根据Doris所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置Doris的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据Doris的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 创建Doris表，并插入10条数据。创建语句如下：

```
CREATE TABLE IF NOT EXISTS dorisdemo
(
  `user_id` varchar(10) NOT NULL,
  `city` varchar(10),
  `age` int,
  `gender` int
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 10;

INSERT INTO dorisdemo VALUES ('user1', 'city1', 20, 1);
INSERT INTO dorisdemo VALUES ('user2', 'city2', 21, 0);
INSERT INTO dorisdemo VALUES ('user3', 'city3', 22, 1);
INSERT INTO dorisdemo VALUES ('user4', 'city4', 23, 0);
INSERT INTO dorisdemo VALUES ('user5', 'city5', 24, 1);
INSERT INTO dorisdemo VALUES ('user6', 'city6', 25, 0);
INSERT INTO dorisdemo VALUES ('user7', 'city7', 26, 1);
INSERT INTO dorisdemo VALUES ('user8', 'city8', 27, 0);
INSERT INTO dorisdemo VALUES ('user9', 'city9', 28, 1);
INSERT INTO dorisdemo VALUES ('user10', 'city10', 29, 0);
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本读取Doris表，并打印。

```
CREATE TABLE dorisDemo (
  `user_id` String NOT NULL,
  `city` String,
  `age` int,
  `gender` int
) with (
```


版权所有 © 华为技术有限公司

```
  'connector' = 'doris',
  'fenodes' = 'FE_IP:PORT,FE_IP:PORT,FE_IP:PORT',
  'table.identifier' = 'demo.dorisdemo',
  'username' = 'dorisUser',
  'password' = 'dorisPassword',
  'doris.request.retries'='3',
  'doris.batch.size' = '100'
);

CREATE TABLE print (
  `user_id` String NOT NULL,
  `city` String,
  `age` int,
  `gender` int
) with (
  'connector' = 'print'
);

insert into print select * from dorisDemo;
```

5. 查看print结果表数据。

```
+I[user5, city5, 24, 1]
+I[user4, city4, 23, 0]
+I[user3, city3, 22, 1]
+I[user10, city10, 29, 0]
+I[user6, city6, 25, 0]
+I[user1, city1, 20, 1]
+I[user9, city9, 28, 1]
+I[user7, city7, 26, 1]
+I[user8, city8, 27, 0]
+I[user2, city2, 21, 0]
```

2.5.5.3 Doris 结果表

功能描述

Flink SQL作业写Doris结果表。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与Doris建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- **如果使用MRS Doris，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《数据湖探索用户指南》中的“**修改主机信息**”章节描述。
- 集群未启用Kerberos认证（普通模式）

  使用**admin**用户连接Doris后，创建具有管理员权限的角色并绑定给用户。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**

- 集群未启用Kerberos认证（普通模式）
- Doris的表名是区分大小写。
- 使用CloudTable的doris时，'fenodes'字段值的端口请用8030，如'xx:8030'。同时安全组请放开端口8030，8040，9030。
- 开启HTTPS后，需要在创建表的**with**子句中添加如下配置参数：
  - **'doris.enable.https' = 'true'**
  - **'doris.ignore.https.ca' = 'true'**
- 请在Flink“作业编辑”页面选择“运行参数配置”，选择“开启Checkpoint”，否则会导致Doris结果表无法写入数据，且写入Doris的延时取决于设置的Checkpoint的间隔时间。

语法格式

```
create table dorisSource (
  attr_name attr_type
  (',' attr_name attr_type)*
 )
with (
  'connector' = 'doris',
  'fenodes' = 'FE_IP:PORT,FE_IP:PORT,FE_IP:PORT',
  'table.identifier' = 'database.table',
  'username' = 'dorisUsername',
  'password' = 'dorisPassword'
);
```

参数说明

通用配置项

参数	默认值	是否必选	参数类型说明
fenodes	--	是	Doris FE ip地址和port, 多实例之间使用逗号分隔。其中port可登录MRS Manager，选择“集群 > 服务 > Doris > 配置”，在搜索框中搜索“http”查看。如果开启https，则搜索“https”。
table.identifier	--	是	Doris 表名，如：db.tbl。
username	--	是	访问 Doris的用户名。
password	--	是	访问 Doris的密码。
sink.label-prefix	""	是	Stream load导入使用的label前缀。2pc场景下要求全局唯一 ，用来保证Flink的EOS语义。
sink.enable-2pc	TRUE	否	是否开启两阶段提交(2pc)，默认为true，保证Exactly-Once语义。
sink.check-interval	10000	否	加载时检查间隔异常。
sink.max-retries	3	否	将记录写入数据库失败时的最大重试次数。

版权所有 © 华为技术有限公司

参数	默认值	是否必选	参数类型说明
sink.buffer-size	256 * 1024	否	缓存流加载数据的缓冲区大小。
sink.buffer-count	3	否	缓存流加载数据的缓冲区计数。
sink.enable-delete	TRUE	否	是否启用删除。此选项需要 Doris 表开启批量删除功能(Doris0.15+版本默认开启)，只支持 Unique 模型。
sink.properties.*	--	否	Stream Load 的导入参数。 例如: 'sink.properties.column_separator' = ',' 定义列分隔符， 'sink.properties.escape_delimiters' = 'true' 特殊字符作为分隔符,'\x01'会被转换为二进制的0x01 JSON格式导入 'sink.properties.format' = 'json' 'sink.properties.read_json_by_line' = 'true'

示例

该示例是从Datagen数据源中生成数据，并将结果写入到Doris结果表中。

1. 参考**增强型跨源连接**，在DLI上根据Doris所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置Doris的安全组，添加入向规则使其对Flink的队列网段放通。分别根据Doris的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。

   参考**测试地址连通性**。
3. 参考MRS Doris使用指南，创建doris表，创建语句如下：

```
CREATE TABLE IF NOT EXISTS dorisdemo
(
  `user_id` varchar(10) NOT NULL,
  `city` varchar(10),
  `age` int,
  `gender` int
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 10
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将Datagen作为数据源，将数据写入到Doris作为结果表中。

```
create table student_datagen_source(
  `user_id` String NOT NULL,
  `city` String,
  `age` int,
  `gender` int
) with (
  'connector' = 'datagen',
```


版权所有 © 华为技术有限公司

```
  'rows-per-second' = '1',
  'fields.user_id.kind' = 'random',
  'fields.user_id.length' = '7',
  'fields.city.kind' = 'random',
  'fields.city.length' = '7'
);


CREATE TABLE dorisDemo (
  `user_id` String NOT NULL,
  `city` String,
  `age` int,
  `gender` int
) with (
  'connector' = 'doris',
  'fenodes' = 'FE_IP:PORT',
  'table.identifier' = 'demo.dorisdemo',
  'username' = 'dorisUser',
  'password' = 'dorisPassword',
  'sink.label-prefix' = 'demo',
  'sink.enable-2pc' = 'true',
  'sink.buffer-count' = '10'
);

insert into dorisDemo select * from student_datagen_source
```

5. 查看doris结果表是否已成功写入数据。

user_id	city	age	gender
50aff04	93406c5	12	1
681a230	1f27d06	16	1
006eff4	3521ded	18	0

2.5.5.4 Doris 维表

功能描述

创建Doris维表用于与输入流连接生成宽表。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- **如果使用MRS Doris，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《数据湖探索用户指南》中的“**修改主机信息**”章节描述。
- 集群未启用Kerberos认证（普通模式）。

  使用**admin**用户连接Doris后，创建具有管理员权限的角色并绑定给用户。

版权所有 © 华为技术有限公司

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 集群未启用Kerberos认证（普通模式）。
- Doris的表名是区分大小写。
- 使用CloudTable的doris时，'fenodes'字段值的端口请用8030，如'xx:8030'。同时安全组请放开端口8030， 8040，9030。
- 开启HTTPS后，需要在创建表的**with**子句中添加如下配置参数：
  - **'doris.enable.https' = 'true'**
  - **'doris.ignore.https.ca' = 'true'**

语法格式

```
create table hbaseSource (
  attr_name attr_type
  (',' attr_name attr_type)*
 )
with (
  'connector' = 'doris',
  'fenodes' = 'FE_IP:PORT,FE_IP:PORT,FE_IP:PORT',
  'table.identifier' = 'database.table',
  'username' = 'dorisUsername',
  'password' = 'dorisPassword'
);
```

参数说明

通用配置项

参数	默认值	是否必选	参数类型说明
fenodes	--	Y	Doris FE ip地址和port, 多实例之间使用逗号分隔。其中port可登录MRS Manager，选择“集群 > 服务 > Doris > 配置”，在搜索框中搜索“http”查看。如果开启https，则搜索“https”。
table.identifier	--	Y	Doris 表名，如：db.tbl
username	--	Y	访问 Doris 的用户名。
password	--	Y	访问 Doris 的密码。
lookup.cache.max-rows	-1L	N	查找缓存的最大行数，超过此值，最旧的行将被删除。 如需启用缓存配置则“cache.max-rows”和“cache.ttl”选项都必须指定。
lookup.cache.ttl	10 s	N	缓存生存时间。

版权所有 © 华为技术有限公司

参数	默认值	是否必选	参数类型说明
lookup.max-retries	3	N	查找数据库失败时的最大重试次数。

示例

该示例是从Doris源表读取数据，并输入到 print connector。

1. 参考**增强型跨源连接**，在DLI上根据Doris所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置Doris和kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据Doris和Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考MRS Doris使用指南，创建doris表，并插入10条数据。创建语句如下：

```
CREATE TABLE IF NOT EXISTS dorisdemo
(
  `user_id` varchar(10) NOT NULL,
  `city` varchar(10),
  `age` int,
  `gender` int
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 10;

INSERT INTO dorisdemo VALUES ('user1', 'city1', 20, 1);
INSERT INTO dorisdemo VALUES ('user2', 'city2', 21, 0);
INSERT INTO dorisdemo VALUES ('user3', 'city3', 22, 1);
INSERT INTO dorisdemo VALUES ('user4', 'city4', 23, 0);
INSERT INTO dorisdemo VALUES ('user5', 'city5', 24, 1);
INSERT INTO dorisdemo VALUES ('user6', 'city6', 25, 0);
INSERT INTO dorisdemo VALUES ('user7', 'city7', 26, 1);
INSERT INTO dorisdemo VALUES ('user8', 'city8', 27, 0);
INSERT INTO dorisdemo VALUES ('user9', 'city9', 28, 1);
INSERT INTO dorisdemo VALUES ('user10', 'city10', 29, 0);
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业模拟从kafka读取数据，并关联doris维表对数据进行打宽，并输出到print。

```
CREATE TABLE ordersSource (
  user_id string,
  user_name string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = 'kafka-topic',
  'properties.bootstrap.servers' = 'kafkaIp:port,kafkaIp:port,kafkaIp:port',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE dorisDemo (
  `user_id` String NOT NULL,
  `city` String,
  `age` int,
  `gender` int
) with (
  'connector' = 'doris',
  'fenodes' = 'FE实例IP地址:端口号',
```


版权所有 © 华为技术有限公司

```
  'table.identifier' = 'demo.dorisdemo',
  'username' = 'dorisUsername',
  'password' = 'dorisPassword',
  'lookup.cache.ttl'='10 m',
  'lookup.cache.max-rows' = '100'
);

CREATE TABLE print (
  user_id string,
  user_name string,
  `city` String,
  `age` int,
  `gender` int
) WITH (
  'connector' = 'print'
);

insert into print
select
  orders.user_id,
  orders.user_name,
  dim.city,
  dim.age,
  dim.gender
from ordersSource orders
left join dorisDemo for system_time as of orders.proctime as dim on orders.user_id = dim.user_id;
```

5. 往kafka数据源写入2条数据。

```
{"user_id": "user1", "user_name": "name1"}
{"user_id": "user2", "user_name": "name2"}
```

6. 查看print结果表数据。

```
+I[user1, name1, city1, 20, 1]
+I[user2, name2, city2, 21, 0]
```

2.5.6 DWS

2.5.6.1 DWS Connector 概述

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DLI将Flink作业从数据仓库服务（DWS）中读取数据。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

DWS的更多信息，请参见**《数据仓库服务管理指南》**。

DLI Flink 1.15版本支持两种DWS Connector方式用于接入GaussDB数据：

- **（推荐使用）DWS服务自研的DWS Connector**：更关注于直接与DWS的性能与交互，用户能够更加灵活便捷的与DWS进行数据的读写操作。

  您可以通过自定义函数（UDF）的方式使用DWS自研的DWS Connector。自定义函数操作请参考**自定义函数**。

  DWS-Connector的使用方法请参考**dws-connector-flink**。

- **（废弃，不推荐使用）DLI服务的DWS Connector**：支持自定义sink和source函数，允许用户根据具体需求实现相应的sink或source函数，以实现特定的数据读写逻辑。

  DLI提供的DWS Connector使用方法请参考**表2-33**

 版权所有 © 华为技术有限公司

表 2-33 DWS Connector 支持类别

类别	操作指导
源表	**DWS源表（不推荐使用）**
结果表	**DWS结果表（不推荐使用）**
维表	**DWS维表（不推荐使用）**

2.5.6.2 DWS 源表（不推荐使用）

功能描述

DLI将Flink作业从数据仓库服务（DWS）中读取数据。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见**《数据仓库服务管理指南》**。

说明

推荐使用DWS服务自研的DWS Connector。

DWS-Connector的使用方法请参考**dws-connector-flink**。

前提条件

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。

  如何创建DWS集群，请参考《数据仓库服务管理指南》中“创建集群”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**

版权所有 © 华为技术有限公司

- with参数中字段只能使用单引号，不能使用双引号。

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector' = 'gaussdb',
  'url' = '',
  'table-name' = '',
  'username' = '',
  'password' = ''
);
```

参数说明

表 2-34 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'gaussdb'。
url	是	无	String	jdbc连接地址。“url”参数中的ip地址请使用DWS的内网地址。 使用gsjdbc4驱动连接时，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。 使用gsjdbc200驱动连接时，格式为：jdbc:gaussdb://${ip}:${port}/${dbName}。
table-name	是	无	String	操作的DWS表名。如果该DWS表在某schema下，则具体可以参考**如果该DWS表在某schema下**的说明。
driver	否	org.postgresql.Driver	String	jdbc连接驱动，默认为: org.postgresql.Driver。 • 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。 • 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。
username	否	无	String	DWS数据库认证用户名，需要和'password'参数一起配置。
password	否	无	String	DWS数据库认证密码，需要和'username'参数一起配置。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
scan.partition.column	否	无	String	用于对输入进行分区的列名。 注意：该参数与scan.partition.lower-bound、scan.partition.upper-bound、 scan.partition.num参数必须同时配置或者同时都不配置。
scan.partition.lower-bound	否	无	Integer	第一个分区的最小值。 与scan.partition.column、scan.partition.upper-bound、 scan.partition.num必须同时配置或者同时都不配置。
scan.partition.upper-bound	否	无	Integer	最后一个分区的最大值。 与scan.partition.column、scan.partition.lower-bound、 scan.partition.num必须同时配置或者同时都不配置。
scan.partition.num	否	无	Integer	分区的个数。 与scan.partition.column、scan.partition.upper-bound、 scan.partition.upper-bound必须同时配置或者同时都不配置。
scan.fetch-size	否	0	Integer	每次从数据库拉取数据的行数。默认值为0，表示不限制。

示例

该示例是从DWS数据源中读取数据，并写入到Print结果表中，其具体步骤参考如下：

1. 在DWS中创建相应的表，表名为dws_order，SQL语句参考如下。

```
create table public.dws_order(
  order_id VARCHAR,
  order_channel VARCHAR,
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR);
```

在DWS中执行以下SQL语句，向dws_order表中插入数据。

```
insert into public.dws_order
  (order_id,
  order_channel,
  order_time,
  pay_amount,
  real_pay,
```


版权所有 © 华为技术有限公司

```
  pay_time,
  user_id,
  user_name,
  area_id) values
  ('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',
'0001', 'Alice', '330106'),
  ('202103251202020001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',
'0002', 'Bob', '330110');
```

2. 参考**增强型跨源连接**，根据DWS所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
3. 设置DWS的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据DWS的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将DWS作为数据源，Print作为结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE dwsSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DWSIP:DWSPort/DWSdbName',
  'table-name' = 'dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword'
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from dwsSource;
```

5. 按照如下操作查看taskmanager.out文件中的数据结果。
   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

```
+I(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24
10:02:03,0001,Alice,330106)
```

 版权所有 © 华为技术有限公司

```
+I(20210325120202001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25
12:03:00,0002,Bob,330110)
```

常见问题

- Q：作业运行失败，运行日志中有如下报错信息，应该怎么解决?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

  A：应考虑是跨源没有绑定，或者跨源没有绑定成功。

  - 参考**增强型跨源连接**章节，重新配置跨源。参考**DLI跨源连接DWS失败**进行问题排查。

- Q：如果该DWS表在某schema下，应该如何配置?

  A：如下示例是使用schema为dbuser2下的表dws_order。

```
CREATE TABLE dwsSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DWSIP:DWSPort/DWSdbName',
  'table-name' = 'dbuser2.dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword'
);
```

2.5.6.3 DWS 结果表（不推荐使用）

功能描述

DLI将Flink作业的输出数据输出到数据仓库服务（DWS）中。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见《数据仓库服务管理指南》。

说明

推荐使用DWS服务自研的DWS Connector。

DWS-Connector的使用方法请参考**dws-connector-flink**。

前提条件

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

版权所有 © 华为技术有限公司

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。如何创建DWS集群，请参考《数据仓库服务管理指南》中“创建集群”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- with参数中字段只能使用单引号，不能使用双引号。
- 若需要使用upsert模式，则必须在DWS结果表和该结果表连接的DWS表都定义主键。
- 若DWS在不同的schema中存在相同名称的表，则在flink opensource sql中需要指定相应的schema。
- 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。该驱动为默认，创建表时可以不填该驱动参数。

  例如，使用gsjdbc4驱动连接、upsert模式写入数据到DWS中。

```
create table dwsSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DwsAddress:DwsPort/DwsDatabase',
  'table-name' = 'car_info',
  'username' = 'DwsUserName',
  'password' = 'DwsPasswrod',
  'write.mode' = 'upsert'
);
```

- 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。

  当DWS表test在名为ads_game_sdk_base的schema下时，可以参考如下样例创建DWS结果表。

```
create table dwsSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector' = 'gaussdb',
  'table-name' = 'ads_game_sdk_base.test',
  'driver' = 'com.huawei.gauss200.jdbc.Driver',
```


版权所有 © 华为技术有限公司

```
  'url' = 'jdbc:gaussdb://DwsAddress:DwsPort/DwsDatabase',
  'username' = 'DwsUserName',
  'password' = 'DwsPasswrod',
  'write.mode' = 'upsert'
);
```

语法格式

说明

DWS结果表中不允许指定所有属性为PRIMARY KEY。

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'gaussdb',
  'url' = '',
  'table-name' = '',
  'driver' = '',
  'username' = '',
  'password' = ''
);
```

参数说明

表 2-35 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定要使用的连接器，这里是'gaussdb'
url	是	无	String	jdbc连接地址 。 使用gsjdbc4驱动连接时，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。 使用gsjdbc200驱动连接时，格式为：jdbc:gaussdb://${ip}:${port}/${dbName}。
table-name	是	无	String	操作的表名。如果该DWS表在某schema下，则格式为：'schema\".\"具体表名'，具体可以参考**常见问题**说明。
driver	否	org.postgresql.Driver	String	jdbc连接驱动，默认为:org.postgresql.Driver。 • 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。 • 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。
username	否	无	String	DWS数据库认证用户名，需要和'password'一起配置

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
password	否	无	String	DWS数据库认证密码，需要和'username'一起配置
write.mode	否	无	String	数据写入模式，支持: copy, insert以及upsert三种。默认值为upsert。 该参数与'primary key'配合使用。 ● 未配置'primary key'时，支持copy及insert两种模式追加写入。 ● 配置'primary key'，支持copy、upsert以及insert三种模式更新写入。 注意：由于dws不支持更新分布列，因而配置的更新主键必须包含dws表中定义的所有分布列。
sink.buffer-flush.max-rows	否	100	Integer	每次写入请求缓存的最大行数。 它能提升写入数据的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。
sink.buffer-flush.interval	否	1s	Duration	刷新缓存的间隔，在这段时间内以异步线程刷新数据。 它能提升写入数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。 注意：'sink.buffer-flush.max-size' 和 'sink.buffer-flush.max-rows' 同时设置为 "0"，并设置刷新缓存的间隔，则以完整的异步处理方式刷新缓存。 格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
sink.max-retries	否	3	Integer	写入最大重试次数。
write.escape-string-value	否	false	Boolean	是否对string类型值进行转义。该参数仅用于write.mode为copy模式下。
key-by-before-sink	否	false	Boolean	在sink算子前是否按指定的主键进行分区。 该参数旨在解决多并发写入的场景下且write.mode为upsert时，如果多个子任务中写入sink的一批数据具有不止一条相同的主键，并且主键相同的这些数据先后顺序不一致，就会导致两个子任务在向DWS根据主键获取行锁时发生互锁的问题。

版权所有 © 华为技术有限公司

示例

该示例是从kafka数据源中读取数据，并以insert模式写入DWS结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，在DLI上根据DWS和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置DWS和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据DWS和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 连接DWS数据库，在DWS中创建相应的表，表名为dws_order，SQL语句参考如下：

```
create table public.dws_order(
  order_id VARCHAR,
  order_channel VARCHAR,
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR);
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将Kafka作业数据源，将DWS作为结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE dwsSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://***DWSAddress:DWSPort/DWSdbName***',
  'table-name' = 'dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = '***DWSUserName***',
```


版权所有 © 华为技术有限公司

```
 'password' = '***DWSPassword***',
 'write.mode' = 'insert'
);

insert into dwsSink select * from kafkaSource;
```

5. 连接Kafka集群，向Kafka中输入以下测试数据。

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

6. 从DWS中使用如下SQL语句查看数据结果。

```
select * from dws_order
```

数据结果参考如下：

```
202103241000000001    webShop    2021-03-24 10:00:00    100.0    100.0    2021-03-24 10:02:03    0001    Alice    330106
```

常见问题

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

  A：应考虑是跨源没有绑定，或者跨源没有绑定成功。

  - 参考**增强型跨源连接**章节，重新配置跨源。参考**DLI跨源连接DWS失败**进行问题排查。

- Q：如果该DWS表在某schema下，则应该如何配置?

  A：当DWS表test在名为ads_game_sdk_base的schema下时，可以参考如下样例中的'table-name'参数配置。

```
CREATE TABLE ads_rpt_game_sdk_realtime_ada_reg_user_pay_mm (
  ddate DATE,
  dmin TIMESTAMP(3),
  game_appkey VARCHAR,
  channel_id VARCHAR,
  pay_user_num_1m bigint,
  pay_amt_1m bigint,
  PRIMARY KEY (ddate, dmin, game_appkey, channel_id) NOT ENFORCED
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://<yourDwsAddress>:<yourDwsPort>/dws_bigdata_db',
  'table-name' = 'ads_game_sdk_base.test',
  'username' = '<yourUsername>',
  'password' = '<yourPassword>',
  'write.mode' = 'upsert'
);
```

- Q：作业运行正常，但是DWS中一直没有数据怎么办?

  A：请分别排查以下场景：

  - 查看jobmanager和taskmanager的日志是否有错误抛出。日志查看操作步骤如下：
    i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
    ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
    iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”或“jobmanager”的文件夹进入，下载获取taskmanager.out和jobmanager.out文件查看结果日志。
  - 验证跨源是否正确绑定且安全组规则已对该队列开放。

 版权所有 © 华为技术有限公司

- 查看所要写入的DWS表是否在多个不同的schema中存在。若存在，则需要在flink作业中指定schema。

2.5.6.4 DWS 维表（不推荐使用）

功能描述

创建DWS表用于与输入流连接，从而生成相应的宽表。

说明

推荐使用DWS服务自研的DWS Connector。

DWS-Connector的使用方法请参考**dws-connector-flink**。

前提条件

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。如何创建DWS集群，请参考《数据仓库服务管理指南》中“**创建集群**”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- with参数中字段只能使用单引号，不能使用双引号。

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector' = 'gaussdb',
  'url' = '',
  'table-name' = '',
  'username' = '',
  'password' = ''
);
```


版权所有 © 华为技术有限公司

参数说明

表 2-36 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'gaussdb'。
url	是	无	String	jdbc连接地址。 使用gsjdbc4驱动连接时，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。 使用gsjdbc200驱动连接时，格式为：jdbc:gaussdb://${ip}:${port}/${dbName}。
table-name	是	无	String	读取数据库中的数据所在的表名。
driver	否	无	String	jdbc连接驱动，默认为:org.postgresql.Driver。 • 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。 • 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。
username	否	无	String	数据库认证用户名，需要和'password'一起配置。
password	否	无	String	数据库认证密码，需要和'username'一起配置。
scan.partition.column	否	无	String	用于对输入进行分区的列名。 与scan.partition.lower-bound、scan.partition.upper-bound、scan.partition.num必须同时存在或者同时不存在。
scan.partition.lower-bound	否	无	Integer	第一个分区的最小值。 与scan.partition.column、scan.partition.upper-bound、scan.partition.num必须同时存在或者同时不存在。
scan.partition.upper-bound	否	无	Integer	最后一个分区的最大值。 与scan.partition.column、scan.partition.lower-bound、scan.partition.num必须同时存在或者同时不存在。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
scan.partition.num	否	无	Integer	分区的个数。 与scan.partition.column、scan.partition.upper-bound、scan.partition.upper-bound必须同时存在或者同时不存在。
scan.fetch-size	否	0	Integer	每次从数据库拉取数据的行数。默认值为0，表示不限制。
scan.auto-commit	否	true	Boolean	设置自动提交标志。 它决定每一个statement是否以事务的方式自动提交。
lookup.cache.max-rows	否	无	Integer	维表配置，缓存的最大行数，超过该值时，最先添加的数据将被标记为过期。 默认表示不使用该配置。
lookup.cache.ttl	否	无	Duration	维表配置，缓存超时时间，超过该时间的数据会被剔除。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。 默认表示不使用该配置。
lookup.max-retries	否	3	Integer	维表配置，数据拉取最大重试次数。

示例

从Kafka源表中读取数据，将DWS表作为维表，并将二者生成的宽表信息写入Kafka结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，在DLI上根据DWS和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置DWS和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据DWS和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 连接DWS数据库实例，在DWS中创建相应的表，作为维表，表名为area_info，SQL语句如下：

```
create table public.area_info(
  area_id VARCHAR,
  area_province_name VARCHAR,
  area_city_name VARCHAR,
  area_county_name VARCHAR,
  area_street_name VARCHAR,
  region_name VARCHAR);
```

4. 连接DWS数据库实例，向DWS维表area_info中插入测试数据，其语句如下：

```
insert into area_info
(area_id, area_province_name, area_city_name, area_county_name, area_street_name, region_name)
values
('330102', 'a1', 'b1', 'c1', 'd1', 'e1'),
```


版权所有 © 华为技术有限公司

```
 ('330106', 'a1', 'b1', 'c2', 'd2', 'e1'),
 ('330108', 'a1', 'b1', 'c3', 'd3', 'e1'),
 ('330110', 'a1', 'b1', 'c4', 'd4', 'e1');
```

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将Kafka作为数据源，DWS作为维表，数据输出到Kafka结果表中。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaSourceTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = 'dws-order',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

--创建地址维表
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) WITH (
  'connector' = 'gaussdb',
  'driver' = 'org.postgresql.Driver',
  'url' = 'jdbc:postgresql://***DwsAddress:DwsPort/DwsDbName***',
  'table-name' = 'area_info',
  'username' = '***DwsUserName***',
  'password' = '***DwsPassword***',
  'lookup.cache.max-rows' = '10000',
  'lookup.cache.ttl' = '2h'
);

--根据地址维表生成详细的包含地址的订单信息宽表
create table order_detail(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) with (
  'connector' = 'kafka',
  'topic' = '***KafkaSinkTopic***',
```


版权所有 © 华为技术有限公司

```
 'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
 'format' = 'json'
);

insert into order_detail
   select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
          area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
          area.area_street_name, area.region_name  from orders
   left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

6. 连接Kafka集群，向kafka中source topic中插入如下测试数据：

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",
"user_name":"Cindy", "area_id":"330108"}
```

7. 连接Kafka集群，读取kafka中sink topic中数据，结果参考如下：

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_ci
ty_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}

{"order_id":"202103251505050001","order_channel":"qqShop","order_time":"2021-03-25
15:05:05","pay_amount":500.0,"real_pay":400.0,"pay_time":"2021-03-25
15:10:00","user_id":"0003","user_name":"Cindy","area_id":"330108","area_province_name":"a1","area_c
ity_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

常见问题

- Q：若Flink作业日志中有如下报错信息，应该怎么解决?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

  A：应考虑是跨源没有绑定，或者跨源没有绑定成功。

  - 参考**增强型跨源连接**章节，重新配置跨源。参考**DLI跨源连接DWS失败**进行问题排查。

- Q：如果该DWS表在某schema下，则应该如何配置?

  A：如下示例是使用schema为dbuser2下的表area_info：

```
--创建地址维表
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) WITH (
 'connector' = 'gaussdb',
 'driver' = 'org.postgresql.Driver',
 'url' = 'jdbc:postgresql://DwsAddress:DwsPort/DwsDbname',
 'table-name' = 'dbuser2.area_info',
```

 版权所有 © 华为技术有限公司

```
 'username' = 'DwsUserName',
 'password' = 'DwsPassword',
 'lookup.cache.max-rows' = '10000',
 'lookup.cache.ttl' = '2h'
);
```

2.5.7 Elasticsearch

功能描述

DLI将Flink作业的输出数据输出到云搜索服务CSS的Elasticsearch 引擎的索引中。

Elasticsearch是基于Lucene的当前流行的企业级搜索服务器，具备分布式多用户的能力。其主要功能包括全文检索、结构化搜索、分析、聚合、高亮显示等。能为用户提供实时搜索、稳定可靠的服务。适用于日志分析、站内搜索等场景。

云搜索服务（Cloud Search Service，简称CSS）为DLI提供托管的分布式搜索引擎服务，完全兼容开源Elasticsearch搜索引擎，支持结构化、非结构化文本的多条件检索、统计、报表。

云搜索服务的更多信息，请参见《云搜索服务用户指南》。

更多具体使用可参考开源社区文档：**Elasticsearch SQL 连接器**。

表 2-37 支持类别

类别	详情
支持表类型	结果表
支持数据格式	JSON

前提条件

- 请务必确保您的账户下已在云搜索服务里创建了集群。如何创建集群请参考《云搜索服务用户指南》中**创建集群**章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- with参数中字段只能使用单引号，不能使用双引号。
- 当前只支持CSS集群7.X及以上版本。
- 如果未开启安全模式，语法中hosts字段值以http开头。
- 如果开启安全模式，未开启https，需要配置用户名username、密码password，且语法中hosts字段值以http开头。
- 如果开启安全模式，开启https，需要配置用户名username、密码password、证书位置certificate。请注意该场景hosts字段值以https开头。

版权所有 © 华为技术有限公司

- CSS集群安全组入向规则必须开启ICMP。
- with参数中字段只能使用单引号，不能使用双引号。
- 数据类型的使用，请参考**Format**章节。

语法格式

```
create table esSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'elasticsearch-7',
  'hosts' = '',
  'index' = ''
);
```

参数说明

表 2-38 Elasticsearch 结果表参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定要使用的连接器，固定为：elasticsearch-7。表示连接到 Elasticsearch 7.x 及更高版本集群。
hosts	是	无	String	Elasticsearch所在集群的主机名，多个以';'间隔。
index	是	无	String	每条记录的 Elasticsearch 索引。可以是静态索引（例如'myIndex'）或动态索引（例如'index-{log_ts|yyyy-MM-dd}'）。更多详细信息，请参见下面的**动态索引**。
username	否	无	String	Elasticsearch所在集群的账号。该账号参数需和密码“password”参数同时配置。
password	否	无	String	Elasticsearch所在集群的密码。该密码参数需和“username”参数同时配置。
document-id.key-delimiter	否	_	String	复合键的分隔符（默认为"_"），例如，指定为"$"将导致文档 ID 为"KEY1$KEY2$KEY3"。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
failure-handler	否	fail	String	对 Elasticsearch 请求失败情况下的失败处理策略。有效策略为： ● fail：如果请求失败并因此导致作业失败，则抛出异常。 ● ignore：忽略失败并放弃请求。 ● retry-rejected：重新添加由于队列容量饱和而失败的请求。 ● 自定义类名称：使用 ActionRequestFailureHandler 的子类进行失败处理。
sink.flush-on-checkpoint	否	true	Boolean	在进行 checkpoint 时是否保证刷出缓冲区中的数据。 如果关闭这一选项，在进行checkpoint时 sink 将不再为所有进行 中的请求等待 Elasticsearch 的执行完成确认。因此，在这种情况下 sink 将不对至少一次的请求的一致性提供任何保证。
sink.bulk-flush.max-actions	否	1000	Interger	每个批量请求的最大缓冲操作数。可以设置'0'为禁用它。
sink.bulk-flush.max-size	否	2mb	MemorySize	每个批量请求的缓冲操作在内存中的最大值。单位必须为 MB。 可以设置为'0'来禁用它。
sink.bulk-flush.interval	否	1s	Duration	flush 缓冲操作的间隔。 可以设置为'0'来禁用它。 注意，'sink.bulk-flush.max-size'和'sink.bulk-flush.max-actions'都设置为'0'的这种 flush 间隔设置允许对缓冲操作进行完全异步处理。
sink.bulk-flush.backoff.strategy	否	DISABLED	String	指定在由于临时请求错误导致任何 flush 操作失败时如何执行重试。有效策略为： ● DISABLED：不执行重试，即第一次请求错误后失败。 ● CONSTANT：等待重试之间的回退延迟。 ● EXPONENTIAL：先等待回退延迟，然后在重试之间指数递增。
sink.bulk-flush.backoff.max-retries	否	无	Integer	最大回退重试次数。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
sink.bulk-flush.backoff.delay	否	无	Duration	每次退避尝试之间的延迟。 对于 CONSTANT 退避策略，该值是每次重试之间的延迟。对于 EXPONENTIAL 退避策略，该值是初始的延迟。
connection.path-prefix	否	无	String	添加到每个REST通信中的前缀字符串，例如， '/v1'。
connection.request-timeout	否	无	Duration	从连接管理器请求连接的超时时间。超时时间必须大于或者等于 0，如果设置为 0 则是无限超时。
connection.timeout	否	无	Duration	建立请求的超时时间 。 超时时间必须大于或者等于 0 ，如果设置为 0 则是无限超时。
socket.timeout	否	无	Duration	等待数据的 socket 的超时时间(SO_TIMEOUT)。超时时间必须大于或者等于 0，如果设置为 0 则是无限超时。
format	否	json	String	Elasticsearch连接器支持指定格式。该格式必须生成有效的 json 文档。默认情况下使用内置'json'格式。 请参考**Format**页面以获取更多详细信息和格式参数。
certificate	否	无	String	Elasticsearch集群的证书在OBS中的位置。 仅在开启安全模式，且开启https下需要配置该参数。 请先在CSS管理控制台下载证书后将证书上传至OBS，该参数配置的是OBS地址。 例如：obs://bucket/path/CloudSearchService.cer

主键处理

Elasticsearch sink 可以根据是否定义了一个主键来确定是在 upsert 模式还是 append 模式下工作。

- 如果定义了主键，Elasticsearch sink 将以upsert模式工作，该模式可以消费包含UPDATE/DELETE消息的查询。
- 如果未定义主键，Elasticsearch sink 将以append模式工作，该模式只能消费包含INSERT消息的查询。

版权所有 © 华为技术有限公司

在Elasticsearch连接器中，主键用于计算Elasticsearch 的文档ID，文档ID为最多512字节且不包含空格的字符串。

Elasticsearch连接器通过使用 document-id.key-delimiter 指定的键分隔符按照 DDL 中定义的顺序连接所有主键字段，为每一行记录生成一个文档ID字符串。 某些类型不允许作为主键字段，因为它们没有对应的字符串表示形式，例如，BYTES，ROW，ARRAY，MAP 等。

如果未指定主键，Elasticsearch 将自动生成文档ID。

动态索引

Elasticsearch sink同时支持静态索引和动态索引。

- 如果您想使用静态索引，则index选项值应为纯字符串，例如 'myusers'，所有记录都将被写入到“myusers”索引中。
- 如果您想使用动态索引，您可以使用 {field_name} 来引用记录中的字段值来动态生成目标索引。
  - 可以使用 '{field_name|date_format_string}' 将 TIMESTAMP/DATE/TIME 类型的字段值转换为 date_format_string 指定的格式。 date_format_string 与 Java 的 **DateTimeFormatter** 兼容。 例如，如果选项值设置为 'myusers-{log_ts|yyyy-MM-dd}'，则 log_ts 字段值为 2020-03-27 12:25:55 的记录将被写入到 “myusers-2020-03-27” 索引中。
  - 可以使用 '{now()|date_format_string}' 将当前的系统时间转换为 date_format_string 指定的格式。now() 对应的时间类型是 TIMESTAMP_WITH_LTZ 。 在将系统时间格式化为字符串时会使用 session 中通过 table.local-time-zone 中配置的时区。 使用 NOW(), now(), CURRENT_TIMESTAMP, current_timestamp 均可以。

> **⚠ 注意**
>
> 使用当前系统时间生成的动态索引时， 对于changelog的流，无法保证同一主键对应的记录能产生相同的索引名, 因此使用基于系统时间的动态索引，只能支持 append only 的流。

示例

该示例是从Kafka数据源中读取数据，并写入到Elasticsearch结果表中（ 本次所使用Elasticsearch版本为7.10.2 ），其具体步骤如下：

1. 参考**增强型跨源连接**，在DLI上根据Elasticsearch和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置Elasticsearch和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据Elasticsearch和Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 登录Elasticsearch集群的Kibana，并选择Dev Tools，输入下列语句并执行，以创建值为orders的index：

```
PUT /orders
{
  "settings": {
    "number_of_shards": 1
  },
```


版权所有 © 华为技术有限公司

```
  "mappings": {
    "properties": {
      "order_id": {
        "type": "text"
      },
      "order_channel": {
        "type": "text"
      },
      "order_time": {
        "type": "text"
      },
      "pay_amount": {
        "type": "double"
      },
      "real_pay": {
        "type": "double"
      },
      "pay_time": {
        "type": "text"
      },
      "user_id": {
        "type": "text"
      },
      "user_name": {
        "type": "text"
      },
      "area_id": {
        "type": "text"
      }
    }
  }
}
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   如下脚本中的加粗参数请根据实际环境修改。

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE elasticsearchSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'elasticsearch-7',
  'hosts' = '***ElasticsearchAddress:ElasticsearchPort***',
  'index' = 'orders'
```


版权所有 © 华为技术有限公司

```
);
insert into elasticsearchSink select * from kafkaSource;
```

5. 连接Kafka集群，向kafka中插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

6. 在Elasticsearch集群的Kibana中输入下述语句并查看相应结果：

```
GET orders/_search
```

```
{
  "took" : 201,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "orders",
        "_type" : "_doc",
        "_id" : "fopyx4sBUuT2wThgYGcp",
        "_score" : 1.0,
        "_source" : {
          "order_id" : "202103241606060001",
          "order_channel" : "appShop",
          "order_time" : "2021-03-24 16:06:06",
          "pay_amount" : 200.0,
          "real_pay" : 180.0,
          "pay_time" : "2021-03-24 16:10:06",
          "user_id" : "0001",
          "user_name" : "Alice",
          "area_id" : "330106"
        }
      },
      {
        "_index" : "orders",
        "_type" : "_doc",
        "_id" : "f4pyx4sBUuT2wThgYGcr",
        "_score" : 1.0,
        "_source" : {
          "order_id" : "202103241000000001",
          "order_channel" : "webShop",
          "order_time" : "2021-03-24 10:00:00",
          "pay_amount" : 100.0,
          "real_pay" : 100.0,
          "pay_time" : "2021-03-24 10:02:03",
          "user_id" : "0001",
          "user_name" : "Alice",
          "area_id" : "330106"
        }
      }
    ]
  }
}
```

 版权所有 © 华为技术有限公司

2.5.8 对象存储 OBS

2.5.8.1 对象存储 OBS 源表

功能描述

文件系统连接器可用于将单个文件或整个目录的数据读取到单个表中。

当使用目录作为source路径时，对目录中的文件进行 无序的读取。更多信息参考**文件系统 SQL 连接器**

语法格式

```
CREATE TABLE sink_table (
   name string,
   num INT,
   p_day string,
   p_hour string
) partitioned by (p_day, p_hour) WITH (
   'connector' = 'filesystem',
   'path' = 'obs://*** ',
   'format' = 'parquet',
   'source.monitor-interval'=''
);
```

参数说明

- **目录监控**

  默认情况下，文件系统连接器是有界的，也就是只会扫描配置路径一遍后就会停止。

  如果需要，可以通过设置 source.monitor-interval 属性来开启目录监控，以便在新文件出现时继续扫描。

键	默认值	类型	描述
source.monitor-interval	无	Duration	设置新文件的监控时间间隔，并且必须设置 > 0 的值。 每个文件都有其路径唯一标识，一旦发现新文件，就会处理一次。 已处理的文件在 source 的整个生命周期内存储在 state 中，因此，source 的 state 在 checkpoint 和 savepoint 时进行保存。 更短的时间间隔意味着文件被更快地发现，但也意味着更频繁地遍历文件系统/对象存储。 如果未设置此配置选项，则提供的路径仅被扫描一次，因此源将是有界的。

- **可用的Metadata**

  以下连接器 metadata 可以在表定义时作为 metadata 列进行访问。所有 metadata 都是只读的。

版权所有 © 华为技术有限公司

键	数据类型	描述
file.path	STRING NOT NULL	输入文件的完整路径。
file.name	STRING NOT NULL	文件名，即距离文件根路径最远的元素。
file.size	STRING NOT NULL	文件的字节数。
file.modification-time	TIMESTAMP_LTZ(3) NOT NULL	文件的修改时间。

示例

从obs表作为数据源读取数据，输出到print connector。

```
CREATE TABLE obs_source(
   name string,
   num INT,
   `file.path` STRING NOT NULL METADATA
) WITH (
   'connector' = 'filesystem',
   'path' = '***obs://demo/sink_parquent_obs***',
   'format' = 'parquet',
   'source.monitor-interval'='1 h'
);


CREATE TABLE print (
   name string,
   num INT,
   path  STRING
) WITH (
   'connector' = 'print'
);

insert into print
select * from obs_source;
```

print 结果：

```
+I[0e72e, 841255524, /spark.db/sink_parquent_obs/compacted-part-fd4d4cc8-8b18-42d5-b522-9b524500fa23-0-0]
+I[53524, -2032270969, /spark.db/sink_parquent_obs/compacted-part-fd4d4cc8-8b18-42d5-b522-9b524500fa23-0-0]
+I[77225, 245599258, /spark.db/sink_parquent_obs/compacted-part-fd4d4cc8-8b18-42d5-b522-9b524500fa23-0-0]
+I[fc202, -545621464, /spark.db/sink_parquent_obs/compacted-part-fd4d4cc8-8b18-42d5-b522-9b524500fa23-0-0]
+I[07e9d, 1511139764, /spark.db/sink_parquent_obs/compacted-part-fd4d4cc8-8b18-42d5-b522-9b524500fa23-0-0]
+I[4e48b, 278014413, /spark.db/sink_parquent_obs/compacted-part-fd4d4cc8-8b18-42d5-b522-9b524500fa23-0-0]
```

2.5.8.2 对象存储 OBS 结果表

功能描述

FileSystem sink用于将数据输出到分布式文件系统HDFS或者对象存储服务OBS等文件系统。适用于数据转储、大数据分析、备份或活跃归档、深度或冷归档等场景。

版权所有 © 华为技术有限公司

考虑到输入流可以是无界的，每个桶中的数据被组织成有限大小的Part文件。完全可以配置为基于时间的方式往桶中写入数据，比如可以设置每个小时的数据写入一个新桶中。即桶中将包含一个小时间隔内接收到的记录。

桶目录中的数据被拆分成多个Part文件。对于相应的接收数据的桶的Sink的每个Subtask，每个桶将至少包含一个Part文件。将根据配置的滚动策略来创建其他Part文件。对于Row Formats默认的策略是根据Part文件大小进行滚动，需要指定文件打开状态最长时间的超时以及文件关闭后的非活动状态的超时时间。对于Bulk Formats在每次创建Checkpoint时进行滚动，并且用户也可以添加基于大小或者时间等的其他条件。更多信息参考**文件系统 SQL 连接器**

说明

- 在STREAMING模式下使用FileSink需要开启Checkpoint功能。Part文件只在Checkpoint成功时生成。如果没有开启Checkpoint功能，文件将永远停留在in-progress或者pending的状态，并且下游系统将不能安全读取该文件数据。
- sink end算子的接受记录数为checkpoint的个数，非实际的发送数据，实际发送数据量请参考streaming-writer或StreamingFileWriter算子的记录数。

注意事项

请在Flink“作业编辑”页面选择“运行参数配置”，选择“开启Checkpoint”，否则会导致FileSystem结果表无法写入数据。

语法格式

```
CREATE TABLE sink_table (
  name string,
  num INT,
  p_day string,
  p_hour string
) partitioned by (p_day, p_hour) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://*** ',
  'format' = 'parquet',
  'auto-compaction' = 'true'
);
```

使用说明

- **滚动策略**

  RollingPolicy 定义了何时关闭给定的In-progress Part文件，并将其转换为Pending状态，然后再转换为Finished状态。 Finished状态的文件，可供查看并且可以保证数据的有效性，在出现故障时不会恢复。

  在 STREAMING模式下，滚动策略结合Checkpoint间隔（到下一个Checkpoint成功时，文件的Pending状态才转换为 Finished 状态），共同控制Part文件对下游readers是否可见以及这些文件的大小和数量。详见滚动策略相关**参数说明**。

- **Part文件生命周期**

  为了在下游使用 FileSink 作为输出，需要了解生成的输出文件的命名和生命周期。

  Part 文件可以处于以下三种状态中的任意一种：

  - **In-progress**：当前正在写入的 Part 文件处于 in-progress 状态
  - **Pending**：由于指定的滚动策略 ）关闭 in-progress 状态文件，并且等待提交
  - **Finished**：流模式(STREAMING)下的成功的 Checkpoint 或者批模式(BATCH)下输入结束，文件的Pending状态转换为 Finished 状态

版权所有 © 华为技术有限公司

只有 Finished 状态下的文件才能被下游安全读取，并且保证不会被修改。

默认的，Part文件命名策略如下:

- In-progress / Pending：part-<uid>-<partFileIndex>.inprogress.uid
- Finished：part-<uid>-<partFileIndex>

当Sink Subtask实例化时，uid是一个分配给 Subtask 的随机ID值。uid不具有容错机制，所以当Subtask从故障恢复时，uid会重新生成。

- **文件合并**

  FileSink 开始支持已经提交Pending文件的合并，从而允许应用设置一个较小的时间周期并且避免生成大量的小文件。

  这一功能开启后，在文件转为Pending状态与文件最终提交之间会进行文件合并。这些Pending状态的文件将首先被提交为一个以.开头的临时文件。这些临时文件随后将会按照用户指定的策略和合并方式进行合并，最终生成合并后的Pending状态的文件。 然后这些文件将被发送给Committer并提交为正式文件，在这之后，原始的临时文件也会被删除掉。

- **分区功能**

  Filesystem sink支持分区功能，通过partitioned by语法根据选择的字段进行分区。示例如下：

```
path
└── datetime=2022-06-25
    └── hour=10
        ├── part-0.parquet
        ├── part-1.parquet
└── datetime=2022-06-26
    └── hour=16
        ├── part-0.parquet
    └── hour=17
        ├── part-0.parquet
```

  分区和文件一样，也需要进行提交，通知下游应用可以安全地读取分区内的文件。Filesystem sink提供多种提交配置策略。

参数说明

表 2-39 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	固定位filesystem。
path	是	无	String	OBS路径。
format	是	无	String	文件格式。 支持csv、parquet格式。

 版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
sink.rolling-policy.file-size	否	128MB	MemorySize	单个part文件最大大小，超过该数值会滚动产生新文件。 **说明** RollingPolicy 定义了何时关闭给定的In-progress Part文件，并将其转换为Pending状态，然后再转换为Finished状态。 Finished状态的文件，可供查看并且可以保证数据的有效性，在出现故障时不会恢复。 在STREAMING模式下，滚动策略结合Checkpoint间隔（到下一个Checkpoint成功时，文件的Pending状态才转换为Finished状态）共同控制Part文件对下游readers是否可见以及这些文件的大小和数量。
sink.rolling-policy.rollover-interval	否	30 min	Duration	单个Part文件处于打开状态的最长时间，超过该时间会滚动产生新文件（默认值30分钟，以避免产生大量小文件）。检查频率是通过sink.rolling-policy.check-interval参数控制的。 **说明** 该参数数字与单位之间必须要有空格。 支持的时间单位包括: d,h,min,s,ms等。 对于bulk格式的文件(parquet、orc、avro)，checkpoint的时间间隔也会控制单个part文件打开的最长时间。
sink.rolling-policy.check-interval	否	1 min	Duration	基于时间的滚动策略的检查间隔。 该属性控制了基于sink.rolling-policy.rollover-interval属性检查文件是否该被滚动的检查频率。
auto-compaction	否	false	Boolean	在流式 sink 中是否开启自动合并功能。数据首先会被写入临时文件。当checkpoint完成后，该checkpoint产生的临时文件会被合并。
compaction.file-size	否	`sink.rolling-policy.file-size`的大小	MemorySize	合并目标文件大小，默认值为滚动文件大小。 **说明** • 只有在同个checkpoint内的文件会被合并，因此最终文件的数量至少等于checkpoint的数量。 • 如果合并时间较长，可能会引起反压，延长checkpoint所需时间。 • 开启该功能后，checkpoint时会产生最终文件，并打开新的文件接收下个checkpoint产生的数据。

版权所有 © 华为技术有限公司

示例 1

使用datagen随机生成数据写入obs的bucketName桶下的fileName目录中。文件生成时间与checkpoint有关，达到30min或128MB时，生成新文件。

```
create table orders(
  name string,
  num INT
) with (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.name.kind' = 'random',
  'fields.name.length' = '5'
);

CREATE TABLE sink_table (
   name string,
   num INT
) WITH (
   'connector' = 'filesystem',
   'path' = 'obs://bucketName/fileName',
   'format' = 'csv',
   'sink.rolling-policy.file-size'='128m',
   'sink.rolling-policy.rollover-interval'='30 min'
);
INSERT into sink_table SELECT * from orders;
```

示例 2

使用datagen随机生成数据写入obs的bucketName桶下的fileName目录中。文件生成时间与checkpoint有关，达到checkpoint间隔或达到100MB时，生成新文件。

```
create table orders(
  name string,
  num INT
) with (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.name.kind' = 'random',
  'fields.name.length' = '5'
);

CREATE TABLE sink_table (
   name string,
   num INT
) WITH (
   'connector' = 'filesystem',
   'path' = 'obs://bucketName/fileName',
   'format' = 'parquet',
   'sink.rolling-policy.file-size'='128m',
   'sink.rolling-policy.rollover-interval'='30 min',
   'auto-compaction'='true',
   'compaction.file-size'='100m'

);
INSERT into sink_table SELECT * from orders;
```

2.5.9 Hbase

HBase连接器支持读取和写入HBase集群。本文档介绍如何使用HBase连接器基于HBase进行SQL查询。

HBase连接器在upsert模式下运行，可以使用 DDL 中定义的主键与外部系统交换更新操作消息。但是主键只能基于HBase的rowkey字段定义。如果没有声明主键，HBase连接器默认取rowkey作为主键。详情可参考**HBase SQL 连接器**

 版权所有 © 华为技术有限公司

2.5.9.1 Hbase 源表

功能描述

创建source流从HBase中获取数据，作为作业的输入数据。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。DLI可以从HBase中读取数据，用于过滤分析、数据转储等场景。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- **如果使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《数据湖探索用户指南》中的“**修改主机信息**”章节描述。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 创建HBase源表的列簇必须定义为ROW类型，字段名对应列簇名（column family），嵌套的字段名对应列限定符名（column qualifier）。

  用户只需在表结构中声明查询中使用的的列簇和列限定符。除了ROW类型的列，剩下的原子数据类型字段（比如，STRING, BIGINT）将被识别为HBase的rowkey，一张表中只能声明一个rowkey。rowkey字段的名字可以是任意的，如果是保留关键字，需要用反引号进行转义。

语法格式

```
create table hbaseSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
  ','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'hbase-2.2',
  'table-name' = '',
  'zookeeper.quorum' = ''
);
```


版权所有 © 华为技术有限公司

参数说明

表 2-40 参数说明

参数	是否必选	默认值	数据	说明
connector	是	无	String	指定使用的连接器，需配置为：hbase-2.2。
table-name	是	无	String	连接的HBase表名。
zookeeper.quorum	是	无	String	格式为：ZookeeperAddress:ZookeeperPort 以MRS Hbase集群为例，该参数的所使用Zookeeper的ip地址和端口号获取方式如下： • 在MRS Manager上，选择“集群 > *待操作的集群名称* > 服务 > ZooKeeper > 实例”，获取ZooKeeper角色实例的IP地址。 • 在MRS Manager上，选择“集群 > 待操作的集群名称 > 服务 > ZooKeeper > 配置 > 全部配置”，搜索参数“clientPort”，获取“clientPort”的参数值即为ZooKeeper的端口。
zookeeper.znode.parent	否	/hbase	String	Zookeeper中的根目录，默认是/hbase。
null-string-literal	否	无	String	当字符串值为null时的存储形式，默认存成"null" 字符串。 HBase的source的编解码将所有数据类型（除字符串外）将null值以空字节来存储。
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。 **创建跨源认证**

数据类型映射

HBase以字节数组存储所有数据，在读和写过程中要序列化和反序列化数据。

Flink的HBase连接器利用HBase（Hadoop) 的工具类org.apache.hadoop.hbase.util.Bytes进行字节数组和Flink数据类型转换。

Flink的HBase连接器将所有数据类型（除字符串外）null值编码成空字节。对于字符串类型，null值的字面值由null-string-literal选项值决定。

版权所有 © 华为技术有限公司

表 2-41 数据类型映射表

Flink数据类型	HBase转换
CHAR/VARCHAR/STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY/VARBINARY	返回 byte[]。
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
SMALLINT	byte[] toBytes(short val) short toShort(byte[] bytes)
INT	byte[] toBytes(int val) int toInt(byte[] bytes)
BIGINT	byte[] toBytes(long val) long toLong(byte[] bytes)
FLOAT	byte[] toBytes(float val) float toFloat(byte[] bytes)
DOUBLE	byte[] toBytes(double val) double toDouble(byte[] bytes)
DATE	从 1970-01-01 00:00:00 UTC 开始的天数，int 值。
TIME	从 1970-01-01 00:00:00 UTC 开始天的毫秒数，int 值。
TIMESTAMP	从 1970-01-01 00:00:00 UTC 开始的毫秒数，long 值。
ARRAY	不支持
MAP/MULTISET	不支持
ROW	不支持

示例

该示例是从HBase数据源中读取数据，并写入到Print结果表中（该示例使用的HBase版本2.2.3）：

版权所有 © 华为技术有限公司

1. 参考**增强型跨源连接**，在DLI上根据HBase所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink作业队列。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置HBase集群的安全组，添加入向规则使其对Flink作业队列网段放通。参考**测试地址连通性**根据HBase的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**MRS HBase的使用**，通过HBase shell在HBase中创建相应的表，表名为order，表中只有一个列簇detail。创建语句参考如下：

```
create 'order', {NAME => 'detail'}
```

4. 在HBase shell中执行下述命令，以插入一条数据：

```
put 'order', '202103241000000001', 'detail:order_channel','webShop'
put 'order', '202103241000000001', 'detail:order_time','2021-03-24 10:00:00'
put 'order', '202103241000000001', 'detail:pay_amount','100.00'
put 'order', '202103241000000001', 'detail:real_pay','100.00'
put 'order', '202103241000000001', 'detail:pay_time','2021-03-24 10:02:03'
put 'order', '202103241000000001', 'detail:user_id','0001'
put 'order', '202103241000000001', 'detail:user_name','Alice'
put 'order', '202103241000000001', 'detail:area_id','330106'
```

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将HBase作为数据源，Print作为结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
create table hbaseSource (
  order_id string,--表示唯一的rowkey
  detail Row( --detail表示列簇
    order_channel string,
    order_time string,
    pay_amount string,
    real_pay string,
    pay_time string,
    user_id string,
    user_name string,
    area_id string),
  primary key (order_id) not enforced
) with (
  'connector' = 'hbase-2.2',
   'table-name' = 'order',
   'zookeeper.quorum' = '***ZookeeperAddress:ZookeeperPort***'
) ;

create table printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount string,
  real_pay string,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) with (
 'connector' = 'print'
);

insert into printSink select order_id,
detail.order_channel,detail.order_time,detail.pay_amount,detail.real_pay,
detail.pay_time,detail.user_id,detail.user_name,detail.area_id from hbaseSource;
```

6. 按照如下方式查看taskmanager.out文件中的数据结果：

   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

 版权所有 © 华为技术有限公司

b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

数据结果参考如下：

```
+I(20210324100000001,webShop,2021-03-24 10:00:00,100.00,100.00,2021-03-24 10:02:03,0001,Alice,330106)
```

常见问题

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

  ```
  java.lang.IllegalArgumentException: offset (0) + length (8) exceed the capacity of the array: 6
  ```

  A：如果HBase表中的数据是以其他方式导入的话，那么其存储是以String格式存储的，所以使用其他的数据格式将会报该错误。需要将Flink创建HBase源表中非string类型的字段的字段类型重新改为String即可。

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

  ```
  org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard from server in 90069ms for connection id 0x0
  ```

  A：跨源未绑定或未绑定成功，或是HBase集群安全组未配置放通DLI队列的网段地址。参考**增强型跨源连接**重新配置跨源，或者HBase集群安全组放通DLI队列的网段地址。

2.5.9.2 Hbase 结果表

功能描述

DLI将作业的输出数据输出到HBase中。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。HBase支持消息数据、报表数据、推荐类数据、风控类数据、日志数据、订单数据等结构化、半结构化的KeyValue数据存储。 利用DLI，用户可方便地将海量数据高速、低时延写入HBase。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- **如果使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《数据湖探索用户指南》中的“**修改主机信息**”章节描述。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

版权所有 © 华为技术有限公司

- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 创建的HBase结果表的列簇必须定义为ROW类型，字段名对应列簇名（column family），嵌套的字段名对应列限定符名（column qualifier）。用户只需在表结构中声明查询中使用的的列簇和列限定符。除了ROW类型的列，剩下的原子数据类型字段（比如，STRING, BIGINT）将被识别为 HBase的rowkey，一张表中只能声明一个rowkey。rowkey字段的名字可以是任意的，如果是保留关键字，需要用反引号。

语法格式

```
create table hbaseSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  ','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
) with (
  'connector' = 'hbase-2.2',
  'table-name' = '',
  'zookeeper.quorum' = ''
);
```

参数说明

表 2-42 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定使用的连接器，固定为：hbase-2.2。
table-name	是	无	String	连接的HBase表名。
zookeeper.quorum	是	无	String	HBase Zookeeper实例信息，格式为：ZookeeperAddress:ZookeeperPort 以MRS Hbase集群为例，该参数的所使用Zookeeper的ip地址和端口号获取方式如下： • 在MRS Manager上，选择“集群 > *待操作的集群名称* > 服务 > ZooKeeper > 实例”，获取ZooKeeper角色实例的IP地址。 • 在MRS Manager上，选择“集群 > 待操作的集群名称 > 服务 > ZooKeeper > 配置 > 全部配置”，搜索参数“clientPort”，获取“clientPort”的参数值即为ZooKeeper的端口。
zookeeper.znode.parent	否	/hbase	String	Zookeeper中的根目录，默认是/hbase。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
null-string-literal	否	null	String	当字符串值为null时的存储形式，默认存成 "null" 字符串。 HBase sink的编解码将所有数据类型（除字符串外）为null值时以空字节来存储。
sink.buffer-flush.max-size	否	2mb	MemorySize	每次写入请求缓存行的最大值。 它能提升写入HBase数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。
sink.buffer-flush.max-rows	否	1000	Integer	每次写入请求缓存的最大行数。 它能提升写入HBase数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。
sink.buffer-flush.interval	否	1s	Duration	刷新缓存的间隔，在这段时间内以异步线程刷新数据。 它能提升写入HBase数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。 注意："sink.buffer-flush.max-size" 和 "sink.buffer-flush.max-rows" 同时设置为 "0"，并设置刷新缓存的间隔，则以完整的异步处理方式刷新缓存。 格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
sink.parallelism	否	无	Integer	为 HBase sink operator 定义并行度。 默认情况下，并行度由框架决定，和连接在一起的上游operator一样。
properties.connector.auth.open	否	无	Boolean	true 代表hbase集群开启了kerberos认证。如果开启了kerberos认证，则必须设置。
properties.connector.kerberos.principal	否	无	String	安全集群的登录用户名。如果开启了kerberos认证，则必须设置。
properties.connector.kerberos.keytab	否	无	String	上传“user.keytab”文件的OBS路径。如果开启了kerberos认证，则必须设置。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
properties.connector.kerberos.krb5	否	无	String	上传“krb5.conf”文件的OBS路径。如果开启了kerberos认证，则必须设置。 注：“krb5.conf”中需移除[libdefaults]下的“renew_lifetime”配置项，否则可能会遇到“Message stream modified (41)”问题。

数据类型映射

HBase以字节数组存储所有数据。在读和写过程中要序列化和反序列化数据。

Flink 的 HBase 连接器利用 HBase（Hadoop) 的工具类 org.apache.hadoop.hbase.util.Bytes进行字节数组和Flink 数据类型转换。

Flink 的 HBase 连接器将所有数据类型（除字符串外）null值编码成空字节。对于字符串类型，null值的字面值由null-string-literal选项值决定。

表 2-43 数据类型映射表

Flink 数据类型	HBase 转换
CHAR / VARCHAR / STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY / VARBINARY	返回 byte[]。
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
SMALLINT	byte[] toBytes(short val) short toShort(byte[] bytes)
INT	byte[] toBytes(int val) int toInt(byte[] bytes)
BIGINT	byte[] toBytes(long val) long toLong(byte[] bytes)
FLOAT	byte[] toBytes(float val) float toFloat(byte[] bytes)

版权所有 © 华为技术有限公司

Flink 数据类型	HBase 转换
DOUBLE	byte[] toBytes(double val) double toDouble(byte[] bytes)
DATE	从 1970-01-01 00:00:00 UTC 开始的天数，int 值。
TIME	从 1970-01-01 00:00:00 UTC 开始天的毫秒数，int 值。
TIMESTAMP	从 1970-01-01 00:00:00 UTC 开始的毫秒数，long 值。
ARRAY	不支持
MAP / MULTISET	不支持
ROW	不支持

示例

该示例是从Kafka数据源中读取数据，并写入到HBase结果表中，其具体步骤如下（该示例中hbase的版本为2.2.3）：

1. 参考**增强型跨源连接**，在DLI上根据HBase和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置HBase和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据HBase和Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**MRS HBase的使用**，通过HBase shell在HBase中创建相应的表，表名为order，表中只有一个列族detail，创建语句如下：

   ```
   create 'order', {NAME => 'detail'}
   ```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将Kafka作为数据源，HBase作为结果表（Rowkey为order_id，列簇名为detail）

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   CREATE TABLE orders (
     order_id string,
     order_channel string,
     order_time string,
     pay_amount double,
     real_pay double,
     pay_time string,
     user_id string,
     user_name string,
     area_id string
   ) WITH (
     'connector' = 'kafka',
     'topic' = '***KafkaTopic***',
     'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
     'properties.group.id' = '***GroupId***',
     'scan.startup.mode' = 'latest-offset',
   ```


版权所有 © 华为技术有限公司

```
  'format' = 'json'
);

create table hbaseSink(
  order_id string,
  detail Row(
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string)
) with (
  'connector' = 'hbase-2.2',
  'table-name' = 'order',
  'zookeeper.quorum' = 'ZookeeperAddress:ZookeeperPort',
  'sink.buffer-flush.max-rows' = '1'
);

insert into hbaseSink select order_id,
Row(order_channel,order_time,pay_amount,real_pay,pay_time,user_id,user_name,area_id) from orders;
```

5. 连接Kafka集群，向Kafka中输入数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

6. 通过HBase shell使用下述语句查看数据结果：

```
 scan 'order'
```

数据结果参考如下：

```
202103241000000001    column=detail:area_id, timestamp=2021-12-16T21:30:37.954, value=330106

202103241000000001    column=detail:order_channel, timestamp=2021-12-16T21:30:37.954, value=webShop

202103241000000001    column=detail:order_time, timestamp=2021-12-16T21:30:37.954, value=2021-03-24 10:00:00

202103241000000001    column=detail:pay_amount, timestamp=2021-12-16T21:30:37.954, value=@Y\x00\x00\x00\x00\x00\x00

202103241000000001    column=detail:pay_time, timestamp=2021-12-16T21:30:37.954, value=2021-03-24 10:02:03

202103241000000001    column=detail:real_pay, timestamp=2021-12-16T21:30:37.954, value=@Y\x00\x00\x00\x00\x00\x00

202103241000000001    column=detail:user_id, timestamp=2021-12-16T21:30:37.954, value=0001

202103241000000001    column=detail:user_name, timestamp=2021-12-16T21:30:37.954, value=Alice

202103241606060001    column=detail:area_id, timestamp=2021-12-16T21:30:44.842, value=330106

202103241606060001    column=detail:order_channel, timestamp=2021-12-16T21:30:44.842, value=appShop

202103241606060001    column=detail:order_time, timestamp=2021-12-16T21:30:44.842, value=2021-03-24 16:06:06

202103241606060001    column=detail:pay_amount, timestamp=2021-12-16T21:30:44.842, value=@i
```

```
\x00\x00\x00\x00\x00\x00

202103241606060001    column=detail:pay_time, timestamp=2021-12-16T21:30:44.842,
value=2021-03-24 16:10:06

202103241606060001    column=detail:real_pay, timestamp=2021-12-16T21:30:44.842, value=@f
\x80\x00\x00\x00\x00\x00

202103241606060001    column=detail:user_id, timestamp=2021-12-16T21:30:44.842, value=0001

202103241606060001    column=detail:user_name, timestamp=2021-12-16T21:30:44.842, value=Alice

202103251202020001    column=detail:area_id, timestamp=2021-12-16T21:30:52.181, value=330110

202103251202020001    column=detail:order_channel, timestamp=2021-12-16T21:30:52.181,
value=miniAppShop

202103251202020001    column=detail:order_time, timestamp=2021-12-16T21:30:52.181,
value=2021-03-25 12:02:02

202103251202020001    column=detail:pay_amount, timestamp=2021-12-16T21:30:52.181, value=@N
\x00\x00\x00\x00\x00\x00

202103251202020001    column=detail:pay_time, timestamp=2021-12-16T21:30:52.181,
value=2021-03-25 12:03:00

202103251202020001    column=detail:real_pay, timestamp=2021-12-16T21:30:52.181, value=@N
\x00\x00\x00\x00\x00\x00

202103251202020001    column=detail:user_id, timestamp=2021-12-16T21:30:52.181, value=0002

202103251202020001    column=detail:user_name, timestamp=2021-12-16T21:30:52.181, value=Bob
```

常见问题

Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

```
org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard from
server in 90069ms for connection id 0x0
```

A：可能是跨源连接未绑定或跨源绑定失败。参考**增强型跨源连接**重新配置跨源，Kafka集群安全组放通DLI队列的网段地址。

2.5.9.3 Hbase 维表

功能描述

创建Hbase维表用于与输入流连接生成宽表。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。
- **如果使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《 数据湖探索用户指南 》中的“**修改主机信息**”章节描述。

 版权所有 © 华为技术有限公司

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 所有 HBase 表的列簇必须定义为ROW类型，字段名对应列簇名（column family），嵌套的字段名对应列限定符名（column qualifier）。用户只需在表结构中声明查询中使用的的列簇和列限定符。除了 ROW 类型的列，剩下的原子数据类型字段（比如，STRING, BIGINT）将被识别为 HBase 的 rowkey，一张表中只能声明一个 rowkey。rowkey 字段的名字可以是任意的，如果是保留关键字，需要用反引号。

语法格式

```
create table hbaseSource (
  attr_name attr_type
  (',' attr_name attr_type)*
 )
with (
  'connector' = 'hbase-2.2',
  'table-name' = '',
  'zookeeper.quorum' = ''
);
```

参数说明

表 2-44 参数说明

参数	是否必选	默认值	参数类型	说明
connector	是	无	String	connector的类型，需配置为：hbase-2.2。
table-name	是	无	String	连接的HBase表名。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	参数类型	说明
zookeeper.quorum	是	无	String	HBase Zookeeper quorum 信息。格式为：ZookeeperAddress:ZookeeperPort。 以MRS Hbase集群为例，该参数的所使用Zookeeper的ip地址和端口号获取方式如下： • 在MRS Manager上，选择“集群 > *待操作的集群名称* > 服务 > ZooKeeper > 实例”，获取ZooKeeper角色实例的IP地址。 • 在MRS Manager上，选择“集群 > 待操作的集群名称 > 服务 > ZooKeeper > 配置 > 全部配置”，搜索参数“clientPort”，获取“clientPort”的参数值即为ZooKeeper的端口。
zookeeper.znode.parent	否	/hbase	String	HBase集群的Zookeeper根目录。
lookup.async	否	false	Boolean	是否设置异步维表。
lookup.cache.max-rows	否	-1	Long	维表配置，缓存的最大行数，超过该值时，缓存中最先添加的条目将被标记为过期。 默认表示不使用该配置。
lookup.cache.ttl	否	-1	Long	维表配置，缓存超时时间，超过该时间的数据会被剔除。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。 默认表示不使用该配置。
lookup.max-retries	否	3	Integer	维表配置，数据拉取最大重试次数。
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。 **创建跨源认证**

数据类型映射

HBase以字节数组存储所有数据。在读和写过程中要序列化和反序列化数据。

Flink的HBase连接器利用HBase（Hadoop）的工具类 org.apache.hadoop.hbase.util.Bytes 进行字节数组和 Flink 数据类型转换。

版权所有 © 华为技术有限公司

Flink的HBase连接器将所有数据类型（除字符串外）null 值编码成空字节。对于字符串类型，null 值的字面值由null-string-literal选项值决定。

表 2-45 数据类型映射表

Flink 数据类型	HBase 转换
CHAR / VARCHAR / STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY / VARBINARY	返回 byte[]。
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
SMALLINT	byte[] toBytes(short val) short toShort(byte[] bytes)
INT	byte[] toBytes(int val) int toInt(byte[] bytes)
BIGINT	byte[] toBytes(long val) long toLong(byte[] bytes)
FLOAT	byte[] toBytes(float val) float toFloat(byte[] bytes)
DOUBLE	byte[] toBytes(double val) double toDouble(byte[] bytes)
DATE	从 1970-01-01 00:00:00 UTC 开始的天数，int 值。
TIME	从 1970-01-01 00:00:00 UTC 开始天的毫秒数，int 值。
TIMESTAMP	从 1970-01-01 00:00:00 UTC 开始的毫秒数，long 值。
ARRAY	不支持
MAP / MULTISET	不支持
ROW	不支持

版权所有 © 华为技术有限公司

示例

该示例是从DMS Kafka数据源中读取数据，将HBase表作为维表，从而生成宽表，并将结果写入到Kafka结果表中，其具体步骤如下（该示例中HBase的版本2.2.3）：

1. 参考**增强型跨源连接**，在DLI上根据HBase和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置HBase和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据HBase和Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**MRS HBase的使用**，通过HBase shell在HBase中创建相应的表，表名为area_info，表中只有一个列族detail，创建语句如下：

   ```
   create 'area_info', {NAME => 'detail'}
   ```

4. 在HBase shell中执行下述语句，插入相应的维表数据：

   ```
   put 'area_info', '330106', 'detail:area_province_name', 'a1'
   put 'area_info', '330106', 'detail:area_city_name', 'b1'
   put 'area_info', '330106', 'detail:area_county_name', 'c2'
   put 'area_info', '330106', 'detail:area_street_name', 'd2'
   put 'area_info', '330106', 'detail:region_name', 'e1'

   put 'area_info', '330110', 'detail:area_province_name', 'a1'
   put 'area_info', '330110', 'detail:area_city_name', 'b1'
   put 'area_info', '330110', 'detail:area_county_name', 'c4'
   put 'area_info', '330110', 'detail:area_street_name', 'd4'
   put 'area_info', '330110', 'detail:region_name', 'e1'
   ```

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将Kafka作为数据源，HBase作为维表，将数据写入到Kafka作为结果表中。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   CREATE TABLE orders (
     order_id string,
     order_channel string,
     order_time string,
     pay_amount double,
     real_pay double,
     pay_time string,
     user_id string,
     user_name string,
     area_id string,
     proctime as Proctime()
   ) WITH (
     'connector' = 'kafka',
     'topic' = '***KafkaSourceTopic***',
     'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
     'properties.group.id' = '***GroupId***',
     'scan.startup.mode' = 'latest-offset',
     'format' = 'json'
   );

   --创建地址维表
   create table area_info (
     area_id string,
     detail row(
       area_province_name string,
       area_city_name string,
       area_county_name string,
       area_street_name string,
       region_name string)
   ) WITH (
   ```


版权所有 © 华为技术有限公司

```
  'connector' = 'hbase-2.2',
  'table-name' = 'area_info',
  'zookeeper.quorum' = '***ZookeeperAddress:ZookeeperPort***',
  'lookup.async' = 'true',
  'lookup.cache.max-rows' = '10000',
  'lookup.cache.ttl' = '2h'
);

--根据地址维表生成详细的包含地址的订单信息宽表
create table order_detail(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) with (
  'connector' = 'kafka',
  'topic' = '<yourSinkTopic>',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'format' = 'json'
);

insert into order_detail
    select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
           area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
           area.area_street_name, area.region_name  from orders
    left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

6. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

7. 连接Kafka集群，在Kafka的sink topic读取数据，结果数据参考如下：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_ci
ty_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_ci
ty_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}
```

 版权所有 © 华为技术有限公司

常见问题

Q：Flink作业日志中有如下报错信息应该怎么解决?

```
org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard from server in 90069ms for connection id 0x0
```

A：可能是跨源连接未绑定或跨源绑定失败。参考**增强型跨源连接**重新配置跨源，Kafka集群安全组放通DLI队列的网段地址。

2.5.10 Hive

2.5.10.1 创建 Hive Catalog

简介

Catalog提供了元数据信息，例如数据库、表、分区、视图以及数据库或其他外部系统中存储的函数和信息。

数据处理最关键的方面之一是管理元数据。 元数据可以是临时的，例如临时表、或者通过TableEnvironment注册的UDF。 元数据也可以是持久化的，例如Hive Metastore中的元数据。Catalog 提供了一个统一的API，用于管理元数据，并使其可以从Table API和SQL查询语句中来访问。详情参考**Apache Flink Catalogs**

功能描述

HiveCatalog有两个用途：作为原生Flink元数据的持久化存储，以及作为读写现有Hive元数据的接口。

Flink 的**Hive 文档**提供了有关设置 HiveCatalog以及访问现有 Hive 元数据的详细信息。详情参考：**Apache Flink Hive Catalog**

HiveCatalog可以用来处理两种类型的表：Hive兼容表和通用表。

- Hive兼容表是以Hive兼容的方式存储的，他们的元数据和实际的数据都在分层存储中。因此，通过flink创建的与hive兼容的表，可以通过hive查询。
- Hive通用表是特定于Flink的。当使用HiveCatalog创建通用表时，只是使用HMS来持久化元数据。虽然这些表对Hive来说是可见的，但Hive不太可能理解元数据。因此，在Hive中使用这样的表会导致未定义的行为。

建议切换到Hive方言来创建Hive兼容表。如果您想用默认的方言创建Hive兼容表，确保在您的表属性中设置'connector'='hive'，否则在HiveCatalog中一个表默认被认为是通用的。如果使用Hive方言，就不需要connector属性。了解**Hive方言**。

注意事项

- 警告Hive Metastore以小写形式存储所有元数据对象名称。
- 如果使用相同名称的目录已经存在，那么将会抛出一个异常。
- Hudi表需要使用hudi catalog。并不适用于hive catalog。
- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

版权所有 © 华为技术有限公司

语法格式

```
CREATE CATALOG myhive
WITH (
    'type' = 'hive',
    'default-database' = 'default',
    'hive-conf-dir' = '/opt/flink/conf'
);

USE CATALOG myhive;
```

参数说明

表 2-46 参数说明

参数	必选	默认值	类型	描述
type	是	无	String	Catalog的类型。 创建HiveCatalog时，该参数必须设置为'hive'。
hive-conf-dir	是	无	String	指向包含 hive-site.xml目录的URI。 该值固定为'hive-conf-dir' = '/opt/flink/conf'
default-database	否	default	String	当一个catalog被设为当前catalog时，所使用的默认当前database。

支持的类型

HiveCatalog支持所有Flink类型的通用表。

对于兼容Hive的表，HiveCatalog需要将Flink数据类型映射到相应的Hive类型。

表 2-47 数据类型映射表

Flink数据类型	Hive数据类型
CHAR(p)	CHAR(p)
VARCHAR(p)	VARCHAR(p)
STRING	STRING
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	LONG
FLOAT	FLOAT
DOUBLE	DOUBLE

版权所有 © 华为技术有限公司

Flink数据类型	Hive数据类型
DECIMAL(p, s)	DECIMAL(p, s)
DATE	DATE
TIMESTAMP(9)	TIMESTAMP
BYTES	BINARY
ARRAY<T>	LIST<T>
MAP	MAP
ROW	STRUCT

说明

- Hive的CHAR(p)最大长度为255。
- Hive的VARCHAR(p)最大长度为65535。
- Hive的MAP只支持原始类型的键，而Flink的MAP可以是任何数据类型。
- Hive的UNION类型不支持。
- Hive的TIMESTAMP总是精度为9，不支持其他精度。另一方面，Hive UDF可以处理精度<=9的TIMESTAMP值。
- Hive不支持Flink的TIMESTAMP_WITH_TIME_ZONE。TIMESTAMP_WITH_LOCAL_TIME_ZONE，和 MULTISET。
- Flink的INTERVAL类型还不能映射到Hive INTERVAL类型。

示例

1. 在Flink OpenSource SQL作业中，创建名为myhive的catalog，并使用它用于管理元数据。

```
CREATE CATALOG myhive WITH (
    'type' = 'hive'
    ,'hive-conf-dir' = '/opt/flink/conf'
);

USE CATALOG myhive;

create table dataGenSource(
  user_id string,
  amount int
) with (
  'connector' = 'datagen',
  'rows-per-second' = '1', --每秒生成一条数据
  'fields.user_id.kind' = 'random', --为字段user_id指定random生成器
  'fields.user_id.length' = '3' --限制user_id长度为3
);

create table printSink(
  user_id string,
  amount int
) with (
  'connector' = 'print'
);

insert into printSink select * from dataGenSource;
```


版权所有 © 华为技术有限公司

2. 查看default数据库中，是否含有dataGenSource、printSink 表。

   说明

   Hive Metastore 以小写形式存储所有元数据对象名称。

   图 2-1 查看 default 数据库

3. 使用名为myhive的catalog中的元数据，新建Flink OpenSource SQL作业。

```
CREATE CATALOG myhive WITH (
    'type' = 'hive'
    ,'hive-conf-dir' = '/opt/flink/conf'
);

USE CATALOG myhive;

insert into printSink select * from dataGenSource;
```

2.5.10.2 Hive 方言

简介

从Flink 1.11.0 开始，在使用Hive方言时，Flink允许用户用Hive语法来编写SQL语句。通过提供与Hive语法的兼容性，改善与Hive的互操作性，并减少用户需要在Flink和Hive之间切换来执行不同语句的情况。详情可参考：**Apache Flink Hive 方言**

功能描述

Flink目前支持两种SQL 方言: default 和 hive。您需要先切换到Hive 方言，然后才能使用Hive语法编写。下面介绍如何使用SQL设置方言。

您可以为执行的每个语句动态切换方言。无需重新启动会话即可使用其他方言。

语法格式

SQL 方言可以通过 table.sql-dialect 属性指定

```
set table.sql-dialect=hive;
```

注意事项

- Hive方言只能用于操作Hive对象，并要求当前Catalog是一个**HiveCatalog** 。
- Hive方言只支持db.table这种两级的标识符，不支持带有Catalog名字的标识符。更多信息请参考**Apache Flink Hive Read & Write**。

版权所有 © 华为技术有限公司

虽然所有Hive版本支持相同的语法，但是一些特定的功能对Hive版本有依赖，请参考**Hive 版本**。

例如，更新数据库位置 只在 Hive-2.4.0 或更高版本支持。

- 执行DML和DQL时应该使用**HiveModule** 。
- 从Flink 1.15版本开始，在使用Hive方言抛出以下异常时，请尝试用opt目录下的flink-table-planner_2.12 jar包来替换lib目录下的flink-table-planner-loader jar包。具体原因请参考 **FLINK-25128**。

2.5.10.3 Hive 源表

简介

Apache Hive 已经成为了数据仓库生态系统中的核心。 它不仅仅是一个用于大数据分析和ETL场景的SQL引擎，同样它也是一个数据管理平台，可用于发现，定义，和演化数据。

Flink与Hive的集成包含两个层面，一是利用了Hive的MetaStore作为持久化的Catalog，二是利用Flink来读写Hive的表。**Overview | Apache Flink**

从Flink 1.11.0开始，在使用 Hive方言时，Flink允许用户用Hive语法来编写SQL语句。通过提供与Hive语法的兼容性，改善与Hive的互操作性，并减少用户需要在Flink和Hive之间切换来执行不同语句的情况。详情可参考：**Apache Flink Hive 方言**

使用HiveCatalog，Apache Flink可以用于统一处理Apache Hive表的BATCH和STREAM。Flink可以作为Hive批处理引擎的更高效的替代方案，或者用于连续读写Hive表，以支持实时数据仓库应用程序。**Apache Flink Hive Read & Write**

功能描述

本节介绍利用Flink来读写Hive的表。Hive源表的定义，以及创建源表时使用的参数和示例代码。详情可参考：**Apache Flink Hive Read & Write**

Flink支持在BATCH 和 STREAMING模式下从Hive读取数据。当作为BATCH应用程序运行时，Flink将在执行查询的时间点对表的状态执行查询。STREAMING读取将持续监控表，并在新数据可用时以增量方式获取新数据。默认情况下，Flink会读取有界的表。

STREAMING读取支持同时使用分区表和非分区表。对于分区表，Flink将监控新分区的生成，并在可用时增量读取它们。对于未分区的表，Flink 会监控文件夹中新文件的生成情况，并增量读取新文件。

前提条件

该场景作业需要建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

版权所有 © 华为技术有限公司

- 数据类型的使用，请参考**Format**章节。
- Hive 方言支持的 DDL 语句，Flink 1.15 当前仅支持使用Hive语法创建OBS表。
  - 使用Hive语法创建OBS表
    - default方言： with 属性中需要设置hive.is-external为true。
    - 使用hive 方言：建表语句需要使用EXTERNAL关键字。
- 开启checkpoint功能。
- 建议切换到Hive方言来创建Hive兼容表。如果您想用默认的方言创建Hive兼容表，确保在您的表属性中设置'connector'='hive'，否则在HiveCatalog中一个表默认被认为是通用的。注意，如果使用Hive方言，就不需要connector属性。
- 监视策略是扫描当前位置路径中的所有目录/文件。许多分区可能会导致性能下降。
- 对未分区表进行流式读取时，要求将每个文件以原子方式写入目标目录。
- 分区表的流式读取要求在 hive 元存储的视图中以原子方式添加每个分区。否则，将使用添加到现有分区的新数据。
- 流式读取不支持 Flink DDL 中的水印语法。这些表不能用于窗口运算符。

语法格式

```
CREATE EXTERNAL TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [column_constraint] [COMMENT col_comment], ... [table_constraint])]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [
    [ROW FORMAT row_format]
    [STORED AS file_format]
  ]
  [LOCATION obs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]

row_format:
  : DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [COLLECTION ITEMS TERMINATED BY
char]
      [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]
      [NULL DEFINED AS char]
  | SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value, ...)]

file_format:
  : SEQUENCEFILE
  | TEXTFILE
  | RCFILE
  | ORC
  | PARQUET
  | AVRO
  | INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

column_constraint:
  : NOT NULL [[ENABLE|DISABLE] [VALIDATE|NOVALIDATE] [RELY|NORELY]]

table_constraint:
  : [CONSTRAINT constraint_name] PRIMARY KEY (col_name, ...) [[ENABLE|DISABLE] [VALIDATE|
NOVALIDATE] [RELY|NORELY]]
```

参数说明

请参考**使用Hive语法创建OBS表**，和**Hive 文档**了解每个DDL语句的语义。

版权所有 © 华为技术有限公司

表 2-48 TBLPROPERTIES 参数说明

参数	是否必选	默认参数	数据类型	说明
streaming-source.enable	否	false	Boolean	是否启用流源。 注意： 请确保每个分区/文件都应该以原子方式写入，否则读取器可能会得到不完整的数据。
streaming-source.partition.include	否	all	String	设置分区读取的选项，支持的选项是 'all' 和 'latest'。默认情况下，该选项为 'all' 。 'all' 表示读取所有分区; 'latest'仅在流式处理 Hive 源表用作 temporal table时才有效。'latest' 表示按 'streaming-source.partition.order'的顺序读取最新的分区。 Flink 支持对最新的 hive 分区进行临时连接，通过启用 'streaming-source.enable'，并将 'streaming-source.partition.include' 设置为 'latest'。同时，用户可以通过配置以下分区相关选项来分配分区比较顺序和数据更新间隔。
streaming-source.monitor-interval	否	None	Duration	连续监视分区/文件的时间间隔。注意：Hive 流式处理读取的默认间隔为'1 min'，Hive 流式处理temporal join的默认间隔为'60 min'，这是因为在当前 Hive 流式处理临时连接实现中，每个 TM 都会访问 Hive metaStore，这可能会对 metaStore 产生压力，这将在未来得到改善。
streaming-source.partition-order	否	partition-name	String	流源的分区顺序，支持 create-time、partition-time 和 partition-name。 create-time 比较分区/文件创建时间，这不是 Hive metaStore 中的分区创建时间，而是文件系统中的文件夹/文件修改时间，如果分区文件夹以某种方式更新，例如将新文件添加到文件夹中，可能会影响数据的使用方式。 partition-time 比较从分区名称中提取的时间。 partition-name 比较分区名称的字母顺序。 对于非分区表，此值应始终为“create-time”。 默认情况下，该值为 partition-name。该选项与已弃用的选项“streaming-source.consume-order”相等。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
streaming-source.consume-start-offset	否	None	String	流式处理消费的起始偏移量。如何解析和比较偏移量取决于您的订单。对于 create-time 和 partition-time，应为时间戳字符串（yyyy-[m]m-[d]d [hh：mm：ss]）。 对于partition-time，将使用分区时间提取器从分区中提取时间。对于 partition-name，是分区名称字符串（例如 pt_year=2020/pt_mon=10/pt_day=01）。

- **Source Parallelism Inference**

  默认情况下，Flink 会根据文件数量和每个文件中的块数来推断其 Hive 读取器的最佳并行度。

  Flink 支持灵活配置并行推理策略。您可以在 TableConfig 中配置以下参数（请注意，这些参数会影响作业的所有源）：

Key	Default	Type	Description
table.exec.hive.infer-source-parallelism	true	Boolean	如果为 true，则根据拆分数推断源并行度。如果为 false，则源的并行度由 config 设置。
table.exec.hive.infer-source-parallelism.max	1000	Integer	设置源运算符的最大推断并行度。

- **Load Partition Splits**

  多线程用于拆分 hive 的分区。您可以使用 table.exec.hive.load-partition-splits.thread-num 来配置线程号。默认值为 3，配置的值应大于 0。

Key	Default	Type	Description
table.exec.hive.load-partition-splits.thread-num	3	Integer	配置的值应大于0。

  SQL 提示可用于将配置应用于 Hive 表，而无需更改其在 Hive 元存储中的定义。**Hints | Apache Flink**

- **Vectorized Optimization upon Read**

  当满足以下条件时，Flink 会自动对 Hive 表进行矢量化读取：

  – 格式：ORC 或 Parquet。

版权所有 © 华为技术有限公司

- 没有复杂数据类型的列，如配置单元类型：List、Map、Struct、Union。

  默认情况下，此功能处于启用状态。可以使用以下配置禁用它。

  ```
  table.exec.hive.fallback-mapred-reader=true
  ```

- **Reading Hive Views**

  Flink 能够从 Hive 定义的视图中读取数据，但存在一些限制：

  - 必须先将 Hive 目录设置为当前目录，然后才能查询视图。这可以通过表 API 中的 tableEnv.useCatalog（ ... ） 或 USE CATALOG ...在 SQL 客户端中。
  - Hive 和 Flink SQL 具有不同的语法，例如不同的保留关键字和文字。确保视图的查询与 Flink 语法兼容。

示例

1. 使用Spark SQL创建Hive语法OBS表，并插入10条数据。模拟数据源。

   ```
   CREATE TABLE IF NOT EXISTS demo.student(
     name STRING,
    score DOUBLE)
   PARTITIONED BY (classNo INT)
   STORED AS PARQUET
   LOCATION 'obs://demo/spark.db/student';

   INSERT INTO demo.student PARTITION(classNo=1) VALUES ('Alice', 90.0), ('Bob', 80.0), ('Charlie',
   70.0), ('David', 60.0), ('Eve', 50.0), ('Frank', 40.0), ('Grace', 30.0), ('Hank', 20.0), ('Ivy', 10.0), ('Jack', 0.0);
   ```

2. 使用Flink SQL展示使用批的方式，从Hive语法OBS表demo.student中读取数据，并打印。需要开启checkpoint。

   ```
   CREATE CATALOG myhive WITH (
       'type' = 'hive',
       'default-database' = 'demo',
        'hive-conf-dir' = '/opt/flink/conf'
   );

   USE CATALOG myhive;

   create table if not exists print (
       name STRING,
       score DOUBLE,
       classNo INT)
   with ('connector' = 'print');

   insert into print
   select * from student;
   ```

   结果（taskmanager的out日志）：

   ```
   +I[Alice, 90.0, 1]
   +I[Bob, 80.0, 1]
   +I[Charlie, 70.0, 1]
   +I[David, 60.0, 1]
   +I[Eve, 50.0, 1]
   +I[Frank, 40.0, 1]
   +I[Grace, 30.0, 1]
   +I[Hank, 20.0, 1]
   +I[Ivy, 10.0, 1]
   +I[Jack, 0.0, 1]
   ```

3. 使用Flink SQL展示使用流的方式，从Hive语法OBS表demo.student中读取数据，并打印。

   ```
   CREATE CATALOG myhive WITH (
       'type' = 'hive' ,
       'default-database' = 'demo',
        'hive-conf-dir' = '/opt/flink/conf'
   );

   USE CATALOG myhive;
   ```

 版权所有 © 华为技术有限公司

```
create table if not exists print (
   name STRING,
   score DOUBLE,
   classNo INT)
with ('connector' = 'print');

insert into print
select * from student /*+ OPTIONS('streaming-source.enable' = 'true', 'streaming-source.monitor-interval' = '3 m') */;
```

上述使用了sql hints功能， SQL 提示可用于将配置应用于 Hive 表，而无需更改其在 Hive 元存储中的定义。详情请参考：**SQL Hints**

2.5.10.4 Hive 结果表

功能描述

本节介绍利用Flink写Hive的表。Hive结果表的定义，以及创建结果表时使用的参数和示例代码。详情可参考：**Apache Flink Hive Read & Write**

Flink 支持在 BATCH 和 STREAMING 模式下从Hive写入数据。

- 当作为BATCH应用程序运行时，Flink将写 Hive表，仅在作业完成时使这些记录可见。BATCH 写入支持追加和覆盖现有表。
- STREAMING 不断写入，向Hive添加新数据，以增量方式提交记录使其可见。用户控制何时/如何触发具有多个属性的提交。流式写入不支持插入覆盖。有关可用配置的完整列表，请参阅流式处理接收器。**Streaming sink**

前提条件

该场景作业需要建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 数据类型的使用，请参考**Format**章节。
- Hive 方言支持的 DDL 语句，Flink 1.15 当前仅支持使用Hive语法创建OBS表。
  - 使用Hive语法创建OBS表
    - default方言： with 属性中需要设置hive.is-external为true。
    - 使用hive 方言：建表语句需要使用EXTERNAL关键字。
- 创建Flink OpenSource SQL作业时，在作业编辑界面配置开启checkpoint功能。

语法格式

```
CREATE EXTERNAL TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [column_constraint] [COMMENT col_comment], ... [table_constraint])]
  [COMMENT table_comment]
```

 版权所有 © 华为技术有限公司

```
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [
    [ROW FORMAT row_format]
    [STORED AS file_format]
  ]
  [LOCATION obs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]

row_format:
  : DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [COLLECTION ITEMS TERMINATED BY
char]
      [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]
      [NULL DEFINED AS char]
  | SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value, ...)]

file_format:
  : SEQUENCEFILE
  | TEXTFILE
  | RCFILE
  | ORC
  | PARQUET
  | AVRO
  | INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

column_constraint:
  : NOT NULL [[ENABLE|DISABLE] [VALIDATE|NOVALIDATE] [RELY|NORELY]]

table_constraint:
  : [CONSTRAINT constraint_name] PRIMARY KEY (col_name, ...) [[ENABLE|DISABLE] [VALIDATE|
NOVALIDATE] [RELY|NORELY]]
```

参数说明

请参考**使用Hive语法创建OBS表**，和**Hive 文档**了解每个DDL语句的语义。

有关可用配置的完整列表，请参阅流式处理接收器。**Streaming sink**

示例

以下示例演示如何使用 Datagen 写入具有分区提交功能的Hive表。

```
CREATE CATALOG myhive WITH (
    'type' = 'hive' ,
    'default-database' = 'demo',
    'hive-conf-dir' = '/opt/flink/conf'
);

USE CATALOG myhive;

SET table.sql-dialect=hive;

-- drop table demo.student_hive_sink;
CREATE EXTERNAL TABLE IF NOT EXISTS demo.student_hive_sink(
  name STRING,
 score DOUBLE)
PARTITIONED BY (classNo INT)
STORED AS PARQUET
LOCATION 'obs://demo/spark.db/student_hive_sink'
TBLPROPERTIES (
   'sink.partition-commit.policy.kind'='metastore,success-file'
 );

SET table.sql-dialect=default;
create table if not exists student_datagen_source(
  name STRING,
  score DOUBLE,
  classNo INT
) with (
```

 版权所有 © 华为技术有限公司

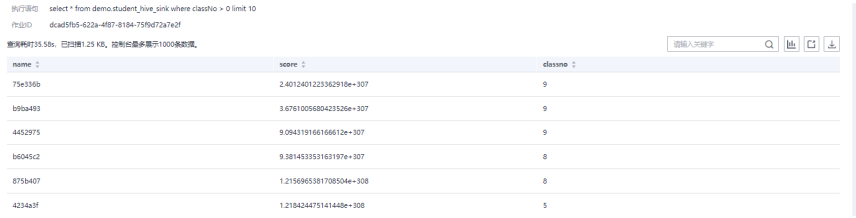
```
 'connector' = 'datagen',
 'rows-per-second' = '1', --每秒生成一条数据
 'fields.name.kind' = 'random', --为字段user_id指定random生成器
 'fields.name.length' = '7', --限制user_id长度为7
 'fields.classNo.kind' ='random',
 'fields.classNo.min' = '1',
 'fields.classNo.max' = '10'
);

insert into student_hive_sink select * from student_datagen_source;
```

使用spark sql进行查询结果表：

```
select * from  demo.student_hive_sink where classNo > 0 limit 10
```

图 2-2 查询结果表



2.5.10.5 Hive 维表

功能描述

您可以将Hive表用作时态表，通过时态连接来关联Hive表。有关时态连接的详细信息，请参阅 **temporal join**。

Flink支持processing-time temporal join Hive Table，processing-time temporal join始终会加入最新版本的时态表。Flink支持分区表和 Hive非分区表的临时连接，对于分区表，Flink 支持自动跟踪Hive表的最新分区。详情可参考：**Apache Flink Hive Read & Write**

注意事项

- Flink目前不支持与Hive表进行基于事件时间event-time的时间关联。
- Temporal Join The Latest Partition 特性，仅在 Flink STREAMING 模式下支持。
- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 数据类型的使用，请参考**Format**章节。
- Hive 方言支持的 DDL 语句，Flink 1.15 当前仅支持使用Hive语法创建OBS表。
  - 使用Hive语法创建OBS表
    - default方言： with 属性中需要设置hive.is-external为true。
    - 使用hive 方言：建表语句需要使用EXTERNAL关键字。
- 创建Flink OpenSource SQL作业时，在作业编辑界面配置开启checkpoint功能。

语法格式、参数说明

请参考**Hive源表**的语法格式和参数说明。

 版权所有 © 华为技术有限公司

2.5.10.6 使用 Temporal join 关联维表的最新分区

功能描述

对于随时间变化的分区表，我们可以将其读取为无界流，如果每个分区包含某个版本的完整数据，则该分区可以被视为时间表的一个版本，时间表的版本保留了分区的数据。Flink支持在处理时间关联中自动跟踪时间表的最新分区（版本）。

最新分区（版本）由 'streaming-source.partition-order' 选项定义。

这是在Flink 流应用作业中将 Hive 表用作维度表的最常见用例。

注意事项

使用Temporal join关联维表的最新分区，仅在Flink STREAMING模式下支持。

示例

下面的示例展示了一个经典的业务流水线，维度表来自 Hive，每天通过批处理流水线作业或 Flink 作业更新一次，kafka流来自实时在线业务数据或日志，需要与维度表连接以扩充流。

1. 使用spark sql 创建 hive obs 外表，并插入数据。

```
CREATE TABLE if not exists dimension_hive_table (
  product_id STRING,
  product_name STRING,
  unit_price DECIMAL(10, 4),
  pv_count BIGINT,
  like_count BIGINT,
  comment_count BIGINT,
  update_time TIMESTAMP,
  update_user STRING
)
STORED AS PARQUET
LOCATION 'obs://demo/spark.db/dimension_hive_table'
PARTITIONED BY (
    create_time    STRING
);
```

```
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_11', 'product_name_11', 1.2345, 100, 50, 20, '2023-11-25 02:10:58', 'update_user_1');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_12', 'product_name_12', 2.3456, 200, 100, 40, '2023-11-25 02:10:58', 'update_user_2');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_13', 'product_name_13', 3.4567, 300, 150, 60, '2023-11-25 02:10:58', 'update_user_3');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_14', 'product_name_14', 4.5678, 400, 200, 80, '2023-11-25 02:10:58', 'update_user_4');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_15', 'product_name_15', 5.6789, 500, 250, 100, '2023-11-25 02:10:58', 'update_user_5');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_16', 'product_name_16', 6.7890, 600, 300, 120, '2023-11-25 02:10:58', 'update_user_6');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_17', 'product_name_17', 7.8901, 700, 350, 140, '2023-11-25 02:10:58', 'update_user_7');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_18', 'product_name_18', 8.9012, 800, 400, 160, '2023-11-25 02:10:58', 'update_user_8');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_19', 'product_name_19', 9.0123, 900, 450, 180, '2023-11-25 02:10:58', 'update_user_9');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_10', 'product_name_10', 10.1234, 1000, 500, 200, '2023-11-25 02:10:58', 'update_user_10');
```

2. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业模拟从kafka读取数据，并关联hive维表对数据进行打宽，并输出到print。

版权所有 © 华为技术有限公司

如下脚本中的加粗参数请根据实际环境修改。

```
CREATE CATALOG myhive WITH (
    'type' = 'hive' ,
    'default-database' = 'demo',
     'hive-conf-dir' = '/opt/flink/conf'
);

USE CATALOG myhive;

CREATE TABLE if not exists ordersSource (
  product_id STRING,
  user_name string,
  proctime as Proctime()
) WITH (
  'connector' = '***kafka***',
  'topic' = '***TOPIC***',
  'properties.bootstrap.servers' = '***KafkaIP:PORT,KafkaIP:PORT,KafkaIP:PORT***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = '***latest-offset***',
  'format' = '***json***'
);

create table if not exists print (
  product_id STRING,
  user_name string,
  product_name STRING,
  unit_price DECIMAL(10, 4),
  pv_count BIGINT,
  like_count BIGINT,
  comment_count BIGINT,
  update_time TIMESTAMP,
  update_user STRING,
  create_time   STRING
) with (
  'connector' = 'print'
);

insert into print
select
  orders.product_id,
  orders.user_name,
  dim.product_name,
  dim.unit_price,
  dim.pv_count,
  dim.like_count,
  dim.comment_count,
  dim.update_time,
  dim.update_user,
  dim.create_time
from ordersSource orders
left join dimension_hive_table /*+ OPTIONS('streaming-source.enable'='true',
   'streaming-source.partition.include' = 'latest', 'streaming-source.monitor-interval' = '10 m') */
   for system_time as of orders.proctime as dim on orders.product_id = dim.product_id;
```

3. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"product_id": "product_id_11", "user_name": "name11"}
{"product_id": "product_id_12", "user_name": "name12"}
```

4. 查看print结果表数据。

```
+I[product_id_11, name11, product_name_11, 1.2345, 100, 50, 20, 2023-11-24T18:10:58,
update_user_1, create_time_1]
+I[product_id_12, name12, product_name_12, 2.3456, 200, 100, 40, 2023-11-24T18:10:58,
update_user_2, create_time_1]
```

5. 模拟向hive 维表，插入新的分区数据

```
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_21', 'product_name_21', 1.2345, 100, 50, 20, '2023-11-25 02:10:58', 'update_user_1');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_22', 'product_name_22', 2.3456, 200, 100, 40, '2023-11-25 02:10:58', 'update_user_2');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
```

```
('product_id_23', 'product_name_23', 3.4567, 300, 150, 60, '2023-11-25 02:10:58', 'update_user_3');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_24', 'product_name_24', 4.5678, 400, 200, 80, '2023-11-25 02:10:58', 'update_user_4');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_25', 'product_name_25', 5.6789, 500, 250, 100, '2023-11-25 02:10:58', 'update_user_5');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_26', 'product_name_26', 6.7890, 600, 300, 120, '2023-11-25 02:10:58', 'update_user_6');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_27', 'product_name_27', 7.8901, 700, 350, 140, '2023-11-25 02:10:58', 'update_user_7');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_28', 'product_name_28', 8.9012, 800, 400, 160, '2023-11-25 02:10:58', 'update_user_8');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_29', 'product_name_29', 9.0123, 900, 450, 180, '2023-11-25 02:10:58', 'update_user_9');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_2') VALUES
('product_id_20', 'product_name_20', 10.1234, 1000, 500, 200, '2023-11-25 02:10:58', 'update_user_10');
```

6. 连接Kafka集群，向Kafka的source topic中插入如下测试数据。关联上一个分区create_time='create_time_1'数据：

```
{"product_id": "product_id_13", "user_name": "name13"}
```

7. 查看print结果表数据。可观察到hive维表中的前一个分区create_time='create_time_1'数据已经被清除

```
+I[product_id_13, name13, null, null, null, null, null, null, null, null]
```

8. 连接Kafka集群，向Kafka的source topic中插入如下测试数据。关联最新分区create_time='create_time_2'数据：

```
{"product_id": "product_id_21", "user_name": "name21"}
```

9. 查看print结果表数据。可观察到hive维表中保存了最新分区create_time='create_time_2'的数据

```
+I[product_id_21, name21, product_name_21, 1.2345, 100, 50, 20, 2023-11-24T18:10:58,
update_user_1, create_time_2]
```

2.5.10.7 使用 Temporal join 关联维表的最新版本

功能描述

对于Hive表，我们可以将其作为有界流读出。在这种情况下，Hive表只能在查询时跟踪其最新版本。最新版本的表保留了Hive表的所有数据。

注意事项

- 每个连接子任务都需要保留自己的Hive表缓存。请确保Hive表可以放入TM任务槽的内存中。
- 建议为streaming-source.monitor-interval（最新分区作为临时表）或lookup.join.cache.ttl（所有分区作为临时表）设置一个相对较大的值。否则，作业容易出现性能问题，避免表更新和重新加载过于频繁。
- 缓存刷新需加载整个Hive表。无法区分新数据和旧数据。

参数说明

在执行与最新的Hive表的时间关联时，Hive表将被缓存到Slot内存中，然后通过键将流中的每条记录与表进行关联，以确定是否找到匹配项。将最新的Hive表用作时间表不需要任何额外的配置。使用以下属性配置Hive表缓存的TTL。在缓存过期后，将重新扫描Hive表以加载最新的数据。

参数	默认值	类型	说明
lookup.join.cache.ttl	60 min	Duration	查找连接中构建表的缓存 TTL（例如 10 分钟）。默认情况下，TTL 为 60 分钟。 该选项仅在查找有界的 hive 表源时有效，如果您使用流式 hive 源作为时态表，请使用 streaming-source.monitor-interval 配置数据更新间隔。

示例

该示例展示了一个经典的业务流水线，维度表来自 Hive，每天通过批处理流水线作业或 Flink 作业更新一次，kafka流来自实时在线业务数据或日志，需要与维度表连接以扩充流。

1. 使用spark sql 创建 hive obs 外表，并插入数据。

```
CREATE TABLE if not exists dimension_hive_table (
  product_id STRING,
  product_name STRING,
  unit_price DECIMAL(10, 4),
  pv_count BIGINT,
  like_count BIGINT,
  comment_count BIGINT,
  update_time TIMESTAMP,
  update_user STRING
)
STORED AS PARQUET
LOCATION 'obs://demo/spark.db/dimension_hive_table'
PARTITIONED BY (
    create_time   STRING
);
```

```
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_11', 'product_name_11', 1.2345, 100, 50, 20, '2023-11-25 02:10:58', 'update_user_1');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_12', 'product_name_12', 2.3456, 200, 100, 40, '2023-11-25 02:10:58', 'update_user_2');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_13', 'product_name_13', 3.4567, 300, 150, 60, '2023-11-25 02:10:58', 'update_user_3');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_14', 'product_name_14', 4.5678, 400, 200, 80, '2023-11-25 02:10:58', 'update_user_4');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_15', 'product_name_15', 5.6789, 500, 250, 100, '2023-11-25 02:10:58', 'update_user_5');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_16', 'product_name_16', 6.7890, 600, 300, 120, '2023-11-25 02:10:58', 'update_user_6');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_17', 'product_name_17', 7.8901, 700, 350, 140, '2023-11-25 02:10:58', 'update_user_7');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_18', 'product_name_18', 8.9012, 800, 400, 160, '2023-11-25 02:10:58', 'update_user_8');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_19', 'product_name_19', 9.0123, 900, 450, 180, '2023-11-25 02:10:58', 'update_user_9');
INSERT INTO dimension_hive_table PARTITION (create_time='create_time_1') VALUES
('product_id_10', 'product_name_10', 10.1234, 1000, 500, 200, '2023-11-25 02:10:58', 'update_user_10');
```

2. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业模拟从kafka读取数据，并关联hive维表对数据进行打宽，并输出到print。

   如下脚本中的加粗参数请根据实际环境修改。

```
CREATE CATALOG myhive WITH (
    'type' = 'hive' ,
    'default-database' = 'demo',
     'hive-conf-dir' = '/opt/flink/conf'
);
```


版权所有 © 华为技术有限公司

```
USE CATALOG myhive;

CREATE TABLE if not exists ordersSource (
  product_id STRING,
  user_name string,
  proctime as Proctime()
) WITH (
  'connector' = '***kafka***',
  'topic' = '***TOPIC***',
  'properties.bootstrap.servers' = '***KafkaIP:PORT,KafkaIP:PORT,KafkaIP:PORT***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = '***latest-offset***',
  'format' = '***json***'
);

create table if not exists print (
  product_id STRING,
  user_name string,
  product_name STRING,
  unit_price DECIMAL(10, 4),
  pv_count BIGINT,
  like_count BIGINT,
  comment_count BIGINT,
  update_time TIMESTAMP,
  update_user STRING,
  create_time   STRING
) with (
  'connector' = 'print'
);

insert into print
select
  orders.product_id,
  orders.user_name,
  dim.product_name,
  dim.unit_price,
  dim.pv_count,
  dim.like_count,
  dim.comment_count,
  dim.update_time,
  dim.update_user,
  dim.create_time
from ordersSource orders
left join dimension_hive_table /*+ OPTIONS('lookup.join.cache.ttl'='60 m') */
   for system_time as of orders.proctime as dim on orders.product_id = dim.product_id;
```

3. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"product_id": "product_id_11", "user_name": "name11"}
{"product_id": "product_id_12", "user_name": "name12"}
```

4. 查看print结果表数据。

```
+I[product_id_11, name11, product_name_11, 1.2345, 100, 50, 20, 2023-11-24T18:10:58,
update_user_1, create_time_1]
+I[product_id_12, name12, product_name_12, 2.3456, 200, 100, 40, 2023-11-24T18:10:58,
update_user_2, create_time_1]
```

2.5.11 Hudi

2.5.11.1 Hudi 源表

功能描述

Flink SQL读取Hudi表数据。

Hudi是一种数据湖的存储格式，在Hadoop文件系统之上提供了更新数据和删除数据的能力以及消费变化数据的能力。支持多种计算引擎，提供IUD接口，在HDFS的数据集上提供了插入更新和增量拉取的功能。

表 2-49 支持类别

类别	详情
支持Flink表类型	源表、结果表。
支持Hudi表类型	MOR表，COW表。
支持读写类型	批量读，批量写，流式读，流式写。

更多具体使用可参考开源社区文档：**Hudi**。

注意事项

- 建议Hudi作为Source表时设置限流

  Hudi表作为Source表时，为防止数据上限超过流量峰值导致作业出现异常，建议设置限流（read.rate.limit），限流上限应该为业务上线压测的峰值。
- 及时对Hudi表进行Compaction，防止Hudi source算子checkpoint完成时间过长

  当Hudi Source算子checkpoint完成时间长时，检查该Hudi表Compaction是否正常。因为当长时间不做Compaction时list性能会变差。
- 流读Hudi MOR表时，建议开启log index特性提升Flink流读性能

  Hudi的Mor表可以通过log index提升读写性能， Sink和Source表添加属性'hoodie.log.index.enabled'='true'
- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table hudiSource (
  attr_name attr_type
  (',' attr_name attr_type)*
 )
with (
  'connector' = 'hudi',
  'path' = 'obs://xx',
  'table.type' = 'xx',
  'hoodie.datasource.write.recordkey.field' = 'xx',
  'write.precombine.field' = 'xx',
  'read.streaming.enabled' = 'xx'
    ...
);
```


版权所有 © 华为技术有限公司

参数说明

说明

当下游消费Hudi过慢，上游写入端会把Hudi文件归档，导致File Not Found问题。设置合理的消费参数避免File Not Found问题。

优化建议：

- 调大read.tasks。
- 如果有限流，调大限流参数。
- 调大上游compaction、archive、clean参数。

表 2-50 参数名称

参数	是否必选	默认值	数据类型	参数说明
connector	是	无	String	读取表类型。需要填写'hudi'
path	是	无	String	表存储的路径。如obs://xx/xx
table.type	是	COPY_ON_WRITE	String	Hudi表类型。 ● MERGE_ON_READ ● COPY_ON_WRITE
hoodie.datasource.write.recordkey.field	是	无	String	表的主键。
write.precombine.field	是	无	String	数据合并字段。
read.tasks	否	4	Integer	读hudi表task并行度。
read.streaming.enabled	是	false	Boolean	设置 true 开启流式增量模式，false批量读。建议值为true
read.streaming.start-commit	否	默认从最新 commit	String	Stream和Batch增量消费，指定“yyyyMMddHHmmss”格式时间的开始消费位置（闭区间）
hoodie.datasource.write.keygenerator.type	否	COMPLEX	Enum	上游表主键生成类型： ● SIMPLE（默认值） ● COMPLEX ● TIMESTAMP ● CUSTOM ● NON_PARTITION ● GLOBAL_DELETE

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
read.streaming.check-interval	否	60	Integer	流读监测上游新提交的周期（秒），流量大时建议使用默认值，默认值：60。
read.end-commit	否	默认到最新 commit	String	Batch增量消费，通过参数“read.streaming.start-commit”指定起始消费位置，通过参数“read.end-commit”指定结束消费位置，为闭区间，即包含起始、结束的Commit，默认到最新Commit。
read.rate.limit	否	0	Integer	流读Hudi的限流速率，单位为每秒的条数。默认值：0，表示不限速。该值为总限速大小，每个算子的限速大小需除以读算子个数（read.tasks）。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
changelog.enabled	否	false	Boolean	是否消费所有变更（包含中间变更）。 参数取值如下： • true：支持消费所有变更。 • false：不消费所有变更，即UPSERT语义，所有的消息仅保证最后一条合并消息，中间的变更可能会被merge掉。 说明 • 只有MOR表支持，在该模式下Hudi会保留消息的所有变更（I/-U/U/D） • 非changelog模式，流读单次的batch数据集会merge中间变更；批读（快照读）会合并所有的中间结果，不管中间状态是否已被写入，都将被忽略。 • 开启changelog.enabled参数后，异步的压缩任务仍然会将中间变更合并成1条数据，所以如果流读消费不够及时，被压缩后只能读到最后一条记录。但是，可以通过调整压缩的频率，预留一定的时间buffer给 reader，比如调整compaction.delta_commits:5和compaction.delta_seconds: 3600压缩参数。

示例 读取 Hudi 表的数据，并将结果输出到 Print 中

1. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE hudiSource (
  order_id STRING PRIMARY KEY NOT ENFORCED,
  order_name STRING,
  order_time TIMESTAMP(3),
```


版权所有 © 华为技术有限公司

```
  order_date String
) WITH (
  'connector' = 'hudi',
  'path' = 'obs://bucket/dir',
  'table.type' = 'MERGE_ON_READ',
  'hoodie.datasource.write.recordkey.field' = 'order_id',
  'write.precombine.field' = 'order_time',
  'read.streaming.enabled' = 'true'
);

create table printSink (
  order_id STRING,
  order_name STRING,
  order_time TIMESTAMP(3),
  order_date String
) with (
  'connector' = 'print'
);

insert into
  printSink
select
  *
from
  hudiSource;
```

2. 该作业提交后，作业状态变成“运行中”，后续您可通过如下操作查看输出结果。
   - 方法一：
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
     iii. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
   - 方法二：如果在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
     iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

2.5.11.2 Hudi 结果表

功能描述

Flink SQL作业写Hudi表。

Hudi是一种数据湖的存储格式，在Hadoop文件系统之上提供了更新数据和删除数据的能力以及消费变化数据的能力。支持多种计算引擎，提供IUD接口，在HDFS的数据集上提供了插入更新和增量拉取的功能。

表 2-51 支持类别

类别	详情
支持Flink表类型	源表、结果表。

 版权所有 © 华为技术有限公司

类别	详情
支持Hudi表类型	MOR表，COW表。
支持读写类型	批量读，批量写，流式读，流式写。

更多具体使用可参考开源社区文档：**Hudi**。

注意事项

- **推荐使用SparkSQL统一建表**
- **表名必须满足Hive格式要求**
  a. 表名必须以字母或下划线开头，不能以数字开头。
  b. 表名只能包含字母、数字、下划线。
  c. 表名长度不能超过128个字符。
  d. 表名中不能包含空格和特殊字符，如冒号、分号、斜杠等。
  e. 表名不区分大小写，但建议使用小写字母。
  f. Hive保留关键字不能作为表名，如select、from、where等。

  示例：

  ```
  my_table、customer_info、sales_data
  ```

  由于作业在触发CheckPoint时才会往Hudi表中写数据，所以需要开启CheckPoint。CheckPoint间隔根据业务需要调整，建议间隔调大。
- 如果CheckPoint间隔太短，数据来不及刷新会导致作业异常；建议CheckPoint间隔为分钟级。
- checkpoint容忍失败次数设置，execution.checkpointing.tolerable-failed-checkpoints。

  Flink On Hudi作业建议设置checkpoint容忍次数多次，如100。
- 若需要使用Hive风格分区，需同时配置如下参数：

  ```
  'hoodie.datasource.write.hive_style_partitioning' = 'true'
  'hive_sync.partition_extractor_class' = 'org.apache.hudi.hive.MultiPartKeysValueExtractor'
  ```
- 默认Hudi写表是Flink状态索引，如果需要使用bucket索引需要在Hudi写表中添加参数：

  ```
  'index.type'='BUCKET',
  'hoodie.bucket.index.num.buckets'='Hudi表中每个分区划分桶的个数',
  'hoodie.bucket.index.hash.field'='recordkey.field'
  ```

  - hoodie.bucket.index.num.buckets：Hudi表中每个分区划分桶的个数，每个分区内的数据通过Hash方式放入每个桶内。建表或第一次写入数据时设置后不能修改，否则更新数据会存在异常。
  - hoodie.bucket.index.hash.field：进行分桶时计算Hash值的字段，必须为主键的子集，默认为Hudi表的主键。该参数不填则默认为recordkey.field。
- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- **Spark离线完成Compaction计划的执行，以及Clean和Archive操作，详见Hudi数据表Compaction规范。**

版权所有 © 华为技术有限公司

Flink作业写MOR表时需要做异步compaction，控制compaction间隔的参数请参考**Hudi官网**。

```
run compaction on <database name>. <table name>;   // 执行Compaction计划
run clean on <database name>. <table name>;        // 执行Clean操作
run archivelog on <database name>.<table name>;    // 执行Archive操作
```

语法格式

```
create table hudiSink (
  attr_name attr_type
  (',' attr_name attr_type)*
 )
with (
  'connector' = 'hudi',
  'path' = 'obs://xx',
  'table.type' = 'MERGE_ON_READ',
  'hoodie.datasource.write.recordkey.field' = 'xx',
  'write.precombine.field' = 'xx',
  'read.streaming.enabled' = 'true'
   ...
);
```

参数说明

参数名称	是否必选	默认值	数据类型	参数描述
connector	是	无	String	读取表类型。需要填写为'hudi'
path	是	无	String	表存储的路径
table.type	是	COPY_ON_WRITE	String	Hudi表类型。 • MERGE_ON_READ • COPY_ON_WRITE
hoodie.datasource.write.recordkey.field	是	无	String	表的主键 • 支持通过PRIMARY KEY语法设置主键字段。 • 支持使用英文逗号（,）分隔多个字段。
hoodie.datasource.write.partitionpath.field	否	无	String	Hudi表的分区字段。无分区表不指定，分区表必须指定
write.precombine.field	是	无	String	数据合并字段 基于此字段的大小来判断消息是否进行更新。 如果您没有设置该参数，则系统默认会按照消息在引擎内部处理的先后顺序进行更新。

版权所有 © 华为技术有限公司

参数名称	是否必选	默认值	数据类型	参数描述
write.payload.class	否	无	String	write.payload.class 参数用于定义数据合并逻辑的方式，具体来说，它指定了在合并更新操作时如何处理相同主键的多条记录。 默认值 OverwriteWithLatestAvroPayload。该策略用于旧记录都会被新记录覆盖。同时也提供了多种预置Payload供用户使用，如 DefaultHoodieRecordPayload、 OverwriteNonDefaultsWithLatestAvroPayload、OverwriteWithLatestAvroPayload及EmptyHoodieRecordPayload。
write.tasks	否	4	Integer	写hudi表task并行度，建议值为4
index.type	否	INMEMORY	String	支持 INMEMORY 或者 BUCKET。默认是 INMEMORY
index.bootstrap.enabled	否	true	Boolean	Flink默认采用的是内存索引（使用Bueckt索引时不配置该项），需要将数据的主键缓存到内存中，保证目标表的数据唯一，因此需要配置该值，否则会导致数据重复，默认值：true。
write.index_bootstrap.tasks	否	环境默认并行度	Integer	“index.bootstrap.enabled”开启后有效，增加任务数提升启动速度，默认值为环境默认并行度。
hoodie.bucket.index.num.buckets	否	5	Integer	Hudi表中每个分区划分桶的个数，每个分区内的数据通过Hash方式放入每个桶内。建表或第一次写入数据时设置后不能修改，否则更新数据会存在异常

版权所有 © 华为技术有限公司

参数名称	是否必选	默认值	数据类型	参数描述
hoodie.bucket.index.hash.field	否	recordkey.field	String	进行分桶时计算Hash值的字段，必须为主键的子集，默认为Hudi表的主键。该参数不填则默认为recordkey.field
index.state.ttl	否	0	Integer	索引数据保存时长，默认值：0（单位：天），表示永久不失效。
compaction.async.enabled	否	false	Boolean	是否开启在线压缩。 ● true：开启 ● false：关闭 建议关闭在线压缩，提升性能。但是调度compaction.schedule.enabled仍然建议开启，之后可通过离线异步压缩，执行阶段性生成的压缩plan。
clean.async.enabled	否	true	Boolean	COW表：设置为true MOR表，且默认开启异步压缩时（compaction.async.enabled = false），需要设置为false，采用异步clean。建议和Compaction放在一起异步去执行
hoodie.archive.automatic	否	true	String	COW表：设置为true MOR表，且默认开启异步压缩时（compaction.async.enabled = false），需要设置为false，采用异步archive。建议和Compaction放在一起异步去执行
compaction.schedule.enabled	否	true	Boolean	是否阶段性生成压缩plan，即使关闭在线压缩的情况下也建议开启
compaction.delta_commits	否	5	Integer	MOR表Compaction计划触发条件。建议值为200。

版权所有 © 华为技术有限公司

参数名称	是否必选	默认值	数据类型	参数描述
compaction.tasks	否	4	Integer	开启在线压缩时，压缩Hudi表task并行度。建议关闭在线压缩，提升性能。
hive_sync.enable	否	false	Boolean	是否启用向Hive同步表信息。 当你需要将Hudi表的元数据同步到 Hive元数据存储中，以便通过Hive查询工具或在DLI控制台的数据管理中访问Hudi表时，需要将此参数设置为 true。 ● true：Hudi 将会把表的元数据（如表结构、分区信息等）同步到 Hive 中。 ● false：不会向Hive同步表信息。 开启向hive同步表信息后会使用catalog相关权限，还需配置访问catalog的委托权限。
hive_sync.mode	否	jdbc	Enum	Hudi表元数据同步到Hive的方式 根据你的环境和需求选择合适的同步模式。 ● **jdbc**：通过JDBC连接到 Hive Server来同步元数据。 ● **hms**：通过Hive Meta Client 直接与Hive Metastore交互来同步元数据。 ● **hiveql**：通过执行Hive QL语句来同步元数据。
hive_sync.table	否	无	String	同步到Hive的表名称，Hudi表的元数据将同步到这个Hive表中
hive_sync.db	否	default	String	同步到Hive的数据库名称，Hudi表的元数据将同步到这个Hive数据库中的表。

版权所有 © 华为技术有限公司

参数名称	是否必选	默认值	数据类型	参数描述
hive_sync.support_timestamp	否	true	Boolean	是否支持时间戳，用于确保Hudi表的元数据同步到Hive时能够正确处理时间戳字段。 建议配置为True。
changelog.enabled	否	false	Boolean	是否消费所有变更（包含中间变更）。 参数取值如下： ● true：支持消费所有变更。 ● false：不消费所有变更，即UPSERT语义，所有的消息仅保证最后一条合并消息，中间的变更可能会被merge掉。 说明 ● 只有MOR表支持，在该模式下Hudi会保留消息的所有变更（I/-U/U/D） ● 非changelog模式，流读单次的batch数据集会merge中间变更；批读（快照读）会合并所有的中间结果，不管中间状态是否已被写入，都将被忽略。 ● 开启changelog.enabled参数后，异步的压缩任务仍然会将中间变更合并成1条数据，所以如果流读消费不够及时，被压缩后只能读到最后一条记录。但是，可以通过调整压缩的频率，预留一定的时间buffer给reader，比如调整compaction.delta_commits:5和compaction.delta_seconds: 3600压缩参数。

版权所有 © 华为技术有限公司

示例 使用 DataGen connector 产生数据，输出到 Hudi 的 MOR 表中(以订单日期作为分区字段)，并使用 HMS 方式同步元数据到 Hive

1. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   create table orderSource (
     order_id STRING,
     order_name STRING,
     order_time TIMESTAMP(3)
   ) with (
     'connector' = 'datagen' ,
     'rows-per-second'='100'
   );

   CREATE TABLE huditest (
     order_id STRING PRIMARY KEY NOT ENFORCED,
     order_name STRING,
     order_time TIMESTAMP(3),
     order_date String
   ) PARTITIONED BY (order_date) WITH (
     'connector' = 'hudi',
     'path' = 'obs://bucket/dir',
     'table.type' = 'MERGE_ON_READ',
     'hoodie.datasource.write.recordkey.field' = 'order_id',
     'write.precombine.field' = 'order_time',
     'hive_sync.enable' = 'true',
     'hive_sync.mode' = 'hms',
     'hive_sync.table' = 'huditest',
     'hive_sync.db' = 'dbtest'
   );

   insert into
     huditest
   select
     order_id, order_name, order_time, DATE_FORMAT(order_time, 'yyyyMMdd')
   from
     orderSource;
   ```

2. 在Spark SQL中执行下述语句，查看写入结果

   ```
   SELECT * FROM dbtest.huditest where order_date = 'xx'
   ```

2.5.12 JDBC

功能描述

JDBC连接器是Flink内置的Connector，提供了对MySQL、PostgreSQL等常见数据库的读写支持。表类型支持源表、结果表和维表。

表 2-52 支持类别

类别	详情
支持表类型	源表、维表、结果表

版权所有 © 华为技术有限公司

前提条件

- 要与实例建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

注意事项

- JDBC结果表如果定义了主键，将以upsert模式与外部系统交换UPDATE/DELETE消息；否则，它将以append模式与外部系统交换消息，不支持消费UPDATE/DELETE消息。
- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**

语法格式

```
create table jbdcTable (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
) with (
  'connector' = 'jdbc',
  'url' = '',
  'table-name' = '',
  'username' = '',
  'password' = ''
);
```

参数说明

表 2-53 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定要使用的连接器，当前固定为'jdbc'。
url	是	无	String	数据库的URL。 • 连接MySQL数据库时，格式为：jdbc:mysql://***MySQLAddress*:*MySQLPort*/**dbName** 。 • 连接PostgreSQL数据库时，格式为：jdbc:postgresql://***PostgreSQLAddress*:*PostgreSQLPort*/**dbName**。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
table-name	是	无	String	读取数据库中的数据所在的表名。
driver	否	无	String	连接数据库所需要的驱动。如果未配置，则会自动通过URL提取。 • MySQL数据库默认驱动为com.mysql.jdbc.Driver。 • PostgreSQL数据库默认驱动为org.postgresql.Driver。
username	否	无	String	数据库认证用户名，需要和'password'一起配置。
password	否	无	String	数据库认证密码，需要和'username'一起配置。
connection.max-retry-timeout	否	60s	Duration	尝试连接数据库服务器最大重试超时时间，不应小于1s。
scan.partition.column	否	无	String	用于对输入进行分区的列名。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.partition.num	否	无	Integer	分区的个数。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.partition.lower-bound	否	无	Integer	第一个分区的最小值。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.partition.upper-bound	否	无	Integer	最后一个分区的最大值。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.fetch-size	否	0	Integer	每次从数据库拉取数据的行数。如果指定为0，则会忽略sql hint。
scan.auto-commit	否	true	Boolean	是否设置自动提交，以确定事务中的每个statement是否自动提交
lookup.cache.max-rows	否	无	Integer	lookup cache的最大行数，如果超过该值，缓存中最先添加的条目将被标记为过期。默认情况下，lookup cache是未开启的。具体请参考**Lookup Cache功能介绍**。
lookup.cache.ttl	否	无	Duration	lookup cache中每一行记录的最大存活时间，如果超过该时间，缓存中最先添加的条目将被标记为过期。 默认情况下，lookup cache是未开启的。具体请参考**Lookup Cache功能介绍**。
lookup.cache.caching-missing-key	否	true	Boolean	是否缓存空查询结果，默认为true。具体请参考**Lookup Cache功能介绍**。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
lookup.max-retries	否	3	Integer	查询数据库失败的最大重试次数。
sink.buffer-flush.max-rows	否	100	Integer	flush前缓存记录的最大值，可以设置为 '0' 来禁用它。
sink.buffer-flush.interval	否	1s	Duration	flush间隔时间，超过该时间后异步线程将flush数据。可以设置为 '0' 来禁用它。如果想完全异步地处理缓存的flush事件，可以将 'sink.buffer-flush.max-rows' 设置为 '0' ，并配置适当的flush时间间隔。
sink.max-retries	否	3	Integer	写入到数据库失败后的最大重试次数。
sink.parallelism	否	无	Integer	用于定义JDBC sink算子的并行度。默认情况下，并行度是由框架决定，即与上游并行度一致。

分区扫描功能介绍

为了加速Source任务实例中的数据读取，Flink为JDBC表提供了分区扫描功能。以下参数定义了从多个任务并行读取时如何对表进行分区。

- scan.partition.column：用于对输入进行分区的列名，该列的数据类型必须是数字，日期或时间戳。
- scan.partition.num: 分区数。
- scan.partition.lower-bound：第一个分区的最小值。
- scan.partition.upper-bound：最后一个分区的最大值。

说明

- **建表时以上扫描分区参数必须同时存在或者同时不存在。**
- scan.partition.lower-bound和scan.partition.upper-bound参数仅用于决定分区步长，而不是用于过滤表中的行，表中的所有行都会被分区并返回。

Lookup Cache 功能介绍

JDBC连接器可以用在时态表关联中作为一个可lookup的维表，当前只支持同步的查找模式。

默认情况下，Lookup cache是未启用的，所有请求都会发送到外部数据库。您可以设置Lookup.cache.max-rows和Lookup.cache.ttl参数来启用。Lookup cache的主要目的是用于提高时态表关联JDBC连接器的性能。

当Lookup cache被启用时，每个进程（即TaskManager）将维护一个缓存。Flink将优先查找缓存，只有当缓存未查找到时才向外部数据库发送请求，并使用返回的数据更新缓存。当缓存命中最大缓存行Lookup.cache.max-rows或当行超过最大存活时间Lookup.cache.ttl时，缓存中最先添加的条目将被标记为过期。缓存中的记录可能不是

版权所有 © 华为技术有限公司

最新的，用户可以将Lookup.cache.ttl设置为一个更小的值以获得更好的刷新数据，但这可能会增加发送到数据库的请求数。所以要做好吞吐量和正确性之间的平衡。

默认情况下，Flink会缓存主键的空查询结果，您可以通过将Lookup.cache.caching-missing-key设置为false来切换行为。

数据类型映射

表 2-54 数据类型映射

MySQL类型	PostgreSQL类型	Flink SQL类型
TINYINT	-	TINYINT
SMALLINT TINYINT UNSIGNED	SMALLINT INT2 SMALLSERIAL SERIAL2	SMALLINT
INT MEDIUMINT SMALLINT UNSIGNED	INTEGER SERIAL	INT
BIGINT INT UNSIGNED	BIGINT BIGSERIAL	BIGINT
BIGINT UNSIGNED	-	DECIMAL(20, 0)
BIGINT	BIGINT	BIGINT
FLOAT	REAL FLOAT4	FLOAT
DOUBLE DOUBLE PRECISION	FLOAT8 DOUBLE PRECISION	DOUBLE
NUMERIC(p, s) DECIMAL(p, s)	NUMERIC(p, s) DECIMAL(p, s)	DECIMAL(p, s)
BOOLEAN TINYINT(1)	BOOLEAN	BOOLEAN
DATE	DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]

版权所有 © 华为技术有限公司

MySQL类型	PostgreSQL类型	Flink SQL类型
CHAR(n) VARCHAR(n) TEXT	CHAR(n) CHARACTER(n) VARCHAR(n) CHARACTER VARYING(n) TEXT	STRING
BINARY VARBINARY BLOB	BYTEA	BYTES
-	ARRAY	ARRAY

示例

- **示例1：使用JDBC作为数据源，Print作为结果表，从RDS MySQL数据库中读取数据，并写入到Print结果表中。**
  a. 参考**增强型跨源连接**，根据RDS MySQL所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
  b. 设置RDS MySQL的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根RDS的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
  c. 登录RDS MySQL，并使用下述命令在flink库下创建orders表，并插入数据。创建数据库的操作可以参考**创建RDS数据库**。

     在flink数据库下创建orders表：

     ```
     CREATE TABLE `flink`.`orders` (
       `order_id` VARCHAR(32) NOT NULL,
       `order_channel` VARCHAR(32) NULL,
       PRIMARY KEY (`order_id`)
     )  ENGINE = InnoDB
       DEFAULT CHARACTER SET = utf8mb4
       COLLATE = utf8mb4_general_ci;
     ```

     插入表数据：

     ```
     insert into orders(
       order_id,
       order_channel
     ) values
       ('1', 'webShop'),
       ('2', 'miniAppShop');
     ```

  d. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

     注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

     认证用的username和password硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**

     ```
     CREATE TABLE jdbcSource (
       order_id string,
     ```


版权所有 © 华为技术有限公司

```
  order_channel string
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://MySQLAddress:MySQLPort/flink',--flink为RDS MySQL创建的数据库名
  'table-name' = 'orders',
  'username' = 'MySQLUsername',
  'password' = 'MySQLPassword',
  'scan.fetch-size' = '10',
  'scan.auto-commit' = 'true'
);

CREATE TABLE printSink (
  order_id string,
  order_channel string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from jdbcSource;
```

e. 查看taskmanager.out文件中的数据结果，数据结果参考如下：

```
+I(1,webShop)
+I(2,miniAppShop)
```

- **示例2：使用DataGen源表发送数据，通过JDBC结果表将数据输出到MySQL数据库中。**

a. 参考**增强型跨源连接**，根据RDS MySQL所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。

b. 设置RDS MySQL的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根RDS的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。

c. 登录RDS MySQL，并使用下述命令在flink库下创建orders表，并插入数据。创建数据库的操作可以参考**创建RDS数据库**。

在flink数据库下创建orders表：

```
CREATE TABLE `flink`.`orders` (
    `order_id` VARCHAR(32) NOT NULL,
    `order_channel` VARCHAR(32) NULL,
    PRIMARY KEY (`order_id`)
)   ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4
    COLLATE = utf8mb4_general_ci;
```

d. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE dataGenSource (
  order_id string,
  order_channel string
) WITH (
  'connector' = 'datagen',
  'fields.order_id.kind' = 'sequence',
  'fields.order_id.start' = '1',
  'fields.order_id.end' = '1000',
  'fields.order_channel.kind' = 'random',
  'fields.order_channel.length' = '5'
);

CREATE TABLE jdbcSink (
  order_id string,
  order_channel string,
  PRIMARY KEY(order_id) NOT ENFORCED
) WITH (
```


版权所有 © 华为技术有限公司

```
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://MySQLAddress:MySQLPort/flink',--其中url中的flink表示MySQL中orders表所在的数据库名
  'table-name' = 'orders',
  'username' = 'MySQLUsername',
  'password' = 'MySQLPassword',
  'sink.buffer-flush.max-rows' = '1'
);

insert into jdbcSink select * from dataGenSource;
```

e. 查看表中数据，在MySQL中执行sql查询语句

```
select * from orders;
```

- **示例3：从DataGen源表中读取数据，将JDBC表作为维表，并将二者生成的表信息写入Print结果表中。**

a. 参考**增强型跨源连接**，根据RDS MySQL所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。

b. 设置RDS MySQL的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根RDS的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。

c. 登录RDS MySQL，并使用下述命令在flink库下创建orders表，并插入数据。创建数据库的操作可以参考**创建RDS数据库**。

在flink数据库下创建orders表：

```
CREATE TABLE `flink`.`orders` (
    `order_id` VARCHAR(32) NOT NULL,
    `order_channel` VARCHAR(32) NULL,
    PRIMARY KEY (`order_id`)
)    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4
    COLLATE = utf8mb4_general_ci;
```

插入表数据：

```
insert into orders(
  order_id,
  order_channel
) values
  ('1', 'webShop'),
  ('2', 'miniAppShop');
```

d. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将DataGen为数据源，JDBC作为维表，数据写入到Print结果表。

注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE dataGenSource (
  order_id string,
  order_time timestamp,
  proctime as Proctime()
) WITH (
  'connector' = 'datagen',
  'fields.order_id.kind' = 'sequence',
  'fields.order_id.start' = '1',
  'fields.order_id.end' = '2'
);

--创建维表
CREATE TABLE jdbcTable (
  order_id string,
  order_channel string
) WITH (
  'connector' = 'jdbc',
```


版权所有 © 华为技术有限公司

```
  'url' = 'jdbc:mysql://JDBCAddress:JDBCPort/flink',--flink为RDS MySQL中orders表所在的数据库名
  'table-name' = 'orders',
  'username' = 'JDBCUserName',
  'password' = 'JDBCPassWord',
  'lookup.cache.max-rows' = '100',
  'lookup.cache.ttl' = '1000',
  'lookup.cache.caching-missing-key' = 'false',
  'lookup.max-retries' = '5'
);

CREATE TABLE printSink (
  order_id string,
  order_time timestamp,
  order_channel string
) WITH (
  'connector' = 'print'
);

insert into
  printSink
SELECT
  dataGenSource.order_id, dataGenSource.order_time, jdbcTable.order_channel
from
  dataGenSource
  left join jdbcTable for system_time as of dataGenSource.proctime on dataGenSource.order_id =
jdbcTable.order_id;
```

e. 查看taskmanager.out文件中的数据结果，数据结果参考如下：

```
+I(1, xxx, webShop)
+I(2, xxx, miniAppShop)
```

常见问题

无

2.5.13 Kafka

功能描述

Kafka 连接器提供从 Kafka topic 中消费和写入数据的能力。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

表 2-55 支持类别

类别	详情
支持表类型	源表、结果表

 版权所有 © 华为技术有限公司

类别	详情
支持数据格式	CSV JSON Apache Avro Confluent Avro Debezium CDC Canal CDC Maxwell CDC OGG CDC Raw

前提条件

- 确保已创建Kafka集群。
- 该场景作业需要运行在DLI的独享队列上，因此要与kafka集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

注意事项

- 更多具体使用可参考开源社区文档：**Apache Kafka SQL 连接器**。
- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- with参数中字段只能使用单引号，不能使用双引号。
- 建表时数据类型的使用请参考**Format**章节。

语法格式

```
create table kafkaSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)
)
with (
  'connector' = 'kafka',
  'topic' = '',
  'properties.bootstrap.servers' = '',
  'properties.group.id' = '',
  'scan.startup.mode' = '',
  'format' = ''
);
```


版权所有 © 华为技术有限公司

源表参数说明

表 2-56 源表参数说明

参数	是否必选	默认值	数据类型	参数说明
connector	是	无	String	指定使用的连接器，Kafka 连接器使用 'kafka'。
topic	否	无	String	当表用作 source 时读取数据的 topic 名。亦支持用分号间隔的 topic 列表，如 'topic-1;topic-2'。 对 source 表而言，'topic' 和 'topic-pattern' 两个选项只能使用其中一个。 当表被用作 sink 时，该配置表示写入的 topic 名。注意 sink 表不支持 topic 列表。
topic-pattern	否	无	String	匹配读取 topic 名称的正则表达式。 在作业开始运行时，所有匹配该正则表达式的 topic 都将被 Kafka consumer 订阅。 注意，对 source 表而言，'topic' 和 'topic-pattern' 两个选项只能使用其中一个。 了解更多请参考**Topic和Partition的探测**。
properties.bootstrap.servers	是	无	String	逗号分隔的 Kafka broker 列表。
properties.group.id	对 source 可选，不适用于 sink	无	String	Kafka source 的消费组 id。如果未指定消费组 ID，则会使用自动生成的 "KafkaSource-{tableIdentifier}" 作为消费组 ID。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
properties.*	否	无	String	设置和传递任意 Kafka 的配置项。 • “properties.”中的后缀名必须匹配在Apache Kafka中定义的配置键。Flink 将移除 "properties." 配置键前缀并将变换后的配置键和值传入底层的 Kafka 客户端。例如，您可以通过 'properties.allow.auto.create.topics' = 'false' 来禁用 topic 的自动创建。 • 某些配置项不支持进行配置，因为 Flink 会覆盖这些配置如 'key.deserializer'和 'value.deserializer'。
format	是	无	String	序列化和反序列化Kafka消息的value的格式。 该配置项和 'value.format' 二者必须其一。 • 关于Kafka消息的消息键和消息体请参考消息键（Key）与消息体（Value）的格式。 • 请参考Format页面以获取更多详细信息和格式参数。
key.format	否	无	String	用来序列化和反序列化 Kafka 消息键（Key）的格式。 • 如果定义了键格式，则配置项 'key.fields' 也是必需的。 否则 Kafka 记录将使用空值作为键。 • 请参考Format页面以获取更多详细信息和格式参数。
key.fields	否	[]	List<String>	表结构中用来配置消息键（Key）格式数据类型的字段列表。 默认情况下该列表为空，因此消息键没有定义。 列表格式为 'field1;field2'。
key.fields-prefix	否	无	String	为所有消息键（Key）格式字段指定自定义前缀，以避免与消息体（Value）格式字段重名。默认情况下前缀为空。 如果定义了前缀，表结构和配置项 'key.fields' 都需要使用带前缀的名称。 当构建消息键格式字段时，前缀会被移除，消息键格式将会使用无前缀的名称。 该配置项要求必须将 'value.fields-include' 配置为 'EXCEPT_KEY'。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
value.format	否	无	String	序列化和反序列化 Kafka 消息体时使用的格式。 • value.format和format参数只能配置其中一个，如果同时配置两个，则会有冲突。 • 请参考**Format**页面以获取更多详细信息和格式参数。
value.fields-include	否	ALL	枚举类型 可选值：[ALL, EXCEPT_KEY]	定义消息体（Value）格式如何处理消息键（Key）字段的策略。 默认情况下，表结构中 'ALL' 即所有的字段都会包含在消息体格式中，即消息键字段在消息键和消息体格式中都会出现。
scan.startup.mode	否	group-offsets	String	Kafka consumer 的启动模式。 取值如下： • earliest-offset：从可能的最早偏移量开始。 • latest-offset：从最末尾偏移量开始。 • group-offsets（默认值）：从 Zookeeper/Kafka 中某个指定的消费组已提交的偏移量开始。 • timestamp：从用户为每个 partition 指定的时间戳开始，时间戳通过scan.startup.timestamp-millis指定。 • specific-offsets：从用户为每个 partition 指定的偏移量开始，位点通过scan.startup.specific-offsets指定。
scan.startup.specific-offsets	否	无	String	在使用 'specific-offsets' 启动模式时为每个 partition 指定 offset，例如 'partition:0,offset:42;partition:1,offset:300'。
scan.startup.timestamp-millis	否	无	Long	在使用 'timestamp' 启动模式时指定启动的时间戳（单位毫秒）。
scan.topic-partition-discovery.interval	否	无	Duration	Consumer 定期探测动态创建的 Kafka topic 和 partition 的时间间隔。

版权所有 © 华为技术有限公司

结果表参数说明

表 2-57 结果表参数说明

参数	是否必选	默认值	数据类型	参数说明
connector	是	无	String	指定使用的连接器，Kafka 连接器使用 'kafka'。
topic	否	无	String	当表用作 source 时读取数据的 topic 名。亦支持用分号间隔的 topic 列表，如 'topic-1;topic-2'。 注意，对 source 表而言，'topic' 和 'topic-pattern' 两个选项只能使用其中一个。 当表被用作 sink 时，该配置表示写入的 topic 名。注意 sink 表不支持 topic 列表。
properties.bootstrap.servers	是	无	String	逗号分隔的 Kafka broker 列表。
properties.*	否	无	String	设置和传递任意 Kafka 的配置项。 • “properties.”中的后缀名必须匹配在Apache Kafka中定义的配置键。Flink 将移除 "properties." 配置键前缀并将变换后的配置键和值传入底层的 Kafka 客户端。例如，您可以通过 'properties.allow.auto.create.topics' = 'false' 来禁用 topic 的自动创建。 • 某些配置项不支持进行配置，因为 Flink 会覆盖这些配置如 'key.deserializer'和 'value.deserializer'。
format	是	无	String	序列化和反序列化Kafka消息的value的格式。**注意：**该配置项和 'value.format' 二者必须其一。 • 关于Kafka消息的消息键和消息体请参考**消息键（Key）与消息体（Value）的格式**。 • 请参考**Format**页面以获取更多详细信息和格式参数。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
key.format	否	无	String	用来序列化和反序列化 Kafka 消息键（Key）的格式。 • 如果定义了键格式，则配置项 'key.fields' 也是必需的。 否则 Kafka 记录将使用空值作为键。 • 请参考**Format**页面以获取更多详细信息和格式参数。
key.fields	否	[]	List<String>	表结构中用来配置消息键（Key）格式数据类型的字段列表。 默认情况下该列表为空，因此消息键没有定义。 列表格式为 'field1;field2'。
key.fields-prefix	否	无	String	为所有消息键（Key）格式字段指定自定义前缀，以避免与消息体（Value）格式字段重名。默认情况下前缀为空。 如果定义了前缀，表结构和配置项 'key.fields' 都需要使用带前缀的名称。 当构建消息键格式字段时，前缀会被移除，消息键格式将会使用无前缀的名称。**注意：**该配置项要求必须将 'value.fields-include' 配置为 'EXCEPT_KEY'。
value.format	否	无	String	序列化和反序列化 Kafka 消息体时使用的格式。 • value.format和format参数只能配置其中一个，如果同时配置两个，则会有冲突。 • 请参考**Format**页面以获取更多详细信息和格式参数。
value.fields-include	否	ALL	枚举类型 可选值：[ALL, EXCEPT_KEY]	定义消息体（Value）格式如何处理消息键（Key）字段的策略。 默认情况下，表结构中 'ALL' 即所有的字段都会包含在消息体格式中，即消息键字段在消息键和消息体格式中都会出现。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
sink.partitioner	否	'default'	String	Flink partition 到 Kafka partition 的分区映射关系，可选值有： ● default：使用 Kafka 默认的分区器对消息进行分区。 ● fixed：每个 Flink partition 最终对应最多一个 Kafka partition。 ● round-robin：Flink partition 按轮循（round-robin）的模式对应到 Kafka partition。只有当未指定消息的消息键时生效。 ● 自定义 FlinkKafkaPartitioner 的子类：例如 'org.mycompany.MyPartitioner'。
sink.semantic	否	at-least-once	String	定义 Kafka sink 的语义。有效值为 'at-least-once'，'exactly-once' 和 'none'。
sink.parallelism	否	无	Integer	定义 Kafka sink 算子的并行度。默认情况下，并行度由框架定义为与上游串联的算子相同。

元数据

您可以在源表中定义元数据，以获取Kafka消息的元数据。

例如，当WITH参数中定义了多个topic时，如果在Kafka源表中定义了元数据，那么Flink读取到的数据就会被标识是从哪个topic中读取的数据。

表 2-58 元数据

Key	数据类型	是否可读(R)写(W)	说明
topic	STRING NOT NULL	R	Kafka 记录的 Topic 名。
partition	INT NOT NULL	R	Kafka 记录的 partition ID。
headers	MAP<STRING, BYTES> NOT NULL	R/W	二进制 Map 类型的 Kafka 记录头（Header）。
leader-epoch	INT NULL	R	Kafka 记录的 Leader epoch（如果可用）。
offset	BIGINT NOT NULL	R	Kafka 记录在 partition 中的 offset。

版权所有 © 华为技术有限公司

Key	数据类型	是否可读(R)写(W)	说明
timestamp	TIMESTAMP(3) WITH LOCAL TIME ZONE NOT NULL	R/W	Kafka 记录的时间戳。
timestamp-type	STRING NOT NULL	R	Kafka 记录的时间戳类型： • NoTimestampType：消息中没有定义时间戳。 • CreateTime：消息产生的时间。 • LogAppendTime：消息被添加到Kafka Broker的时间。

消息键（Key）与消息体（Value）的格式

Kafka消息的消息键和消息体部分都可以使用某种格式来序列化或反序列化成二进制数据。

- **消息体格式**

  由于 Kafka 消息中消息键是可选的，以下语句将使用消息体格式读取和写入消息，但不使用消息键格式。 'format' 选项与 'value.format' 意义相同。 所有的格式配置使用格式识别符作为前缀。

  ```
  CREATE TABLE KafkaTable (
    `ts` TIMESTAMP(3) METADATA FROM 'timestamp',
    `user_id` BIGINT,
    `item_id` BIGINT,
    `behavior` STRING
  ) WITH (
    'connector' = 'kafka',
    ...

    'format' = 'json',
    'json.ignore-parse-errors' = 'true'
  )
  ```

  消息体格式将配置为以下的数据类型：

  ```
  ROW<`user_id` BIGINT, `item_id` BIGINT, `behavior` STRING>
  ```

- **消息键和消息体格式**

  以下示例展示了如何配置和使用消息键和消息体格式。 格式配置使用 'key' 或 'value' 加上格式识别符作为前缀。

  ```
  CREATE TABLE KafkaTable (
    `ts` TIMESTAMP(3) METADATA FROM 'timestamp',
    `user_id` BIGINT,
    `item_id` BIGINT,
    `behavior` STRING
  ) WITH (
    'connector' = 'kafka',
    ...

    'key.format' = 'json',
    'key.json.ignore-parse-errors' = 'true',
    'key.fields' = 'user_id;item_id',
  ```


版权所有 © 华为技术有限公司

```
  'value.format' = 'json',
  'value.json.fail-on-missing-field' = 'false',
  'value.fields-include' = 'ALL'
)
```

消息键格式包含了在 'key.fields' 中列出的字段（使用 ';' 分隔）和字段顺序。 因此将配置为以下的数据类型：

```
ROW<`user_id` BIGINT, `item_id` BIGINT>
```

由于消息体格式配置为 'value.fields-include' = 'ALL'，所以消息键字段也会出现在消息体格式的数据类型中：

```
ROW<`user_id` BIGINT, `item_id` BIGINT, `behavior` STRING>
```

- **重名的格式字段**

  如果消息键字段和消息体字段重名，连接器无法根据表结构信息将这些列区分开。 'key.fields-prefix' 配置项可以在表结构中为消息键字段指定一个唯一名称，并在配置消息键格式的时候保留原名。

  以下示例展示了在消息键和消息体中同时包含 version 字段的情况：

```
CREATE TABLE KafkaTable (
  `k_version` INT,
  `k_user_id` BIGINT,
  `k_item_id` BIGINT,
  `version` INT,
  `behavior` STRING
) WITH (
  'connector' = 'kafka',
  ...

  'key.format' = 'json',
  'key.fields-prefix' = 'k_',
  'key.fields' = 'k_version;k_user_id;k_item_id',

  'value.format' = 'json',
  'value.fields-include' = 'EXCEPT_KEY'
)
```

消息体格式必须配置为 'EXCEPT_KEY' 模式。格式将被配置为以下的数据类型：

```
消息键格式：
ROW<`version` INT, `user_id` BIGINT, `item_id` BIGINT>

消息体格式：
ROW<`version` INT, `behavior` STRING>
```

Topic 和 Partition 的探测

topic 和 topic-pattern 配置项决定了 source 消费的 topic 或 topic 的匹配规则。topic 配置项可接受使用分号间隔的 topic 列表，例如 topic-1;topic-2。 topic-pattern 配置项使用正则表达式来探测匹配的 topic。例如 topic-pattern 设置为 test-topic-[0-9]，则在作业启动时，所有匹配该正则表达式的 topic（以 test-topic- 开头，以一位数字结尾）都将被 consumer 订阅。

为允许 consumer 在作业启动之后探测到动态创建的 topic，请将 scan.topic-partition-discovery.interval 配置为一个非负值。这将使 consumer 能够探测匹配名称规则的 topic 中新的 partition。

说明

topic列表和topic匹配规则只适用于 source。对于sink端，Flink目前只支持单一topic。

 版权所有 © 华为技术有限公司

示例 1：读取 CSV 格式 DMS Kafka 的元数据，输出到 Kafka sink 中（适用于 Kafka 集群未开启 SASL_SSL 场景）

1. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource(
  `topic` String metadata virtual,
  `partition` int metadata virtual,
  `headers` MAP<STRING, BYTES> metadata virtual,
  `leader-epoch` INT metadata virtual,
  `offset` bigint metadata virtual,
  `timestamp-type` string metadata virtual,
  `event_time` TIMESTAMP(3) metadata FROM 'timestamp',
  `message` string
) WITH (
  'connector' = 'kafka',
  'topic' = '***SourceKafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv',
  'csv.field-delimiter' = '\u0001',
  'csv.quote-character' = '"'
);

CREATE TABLE kafkaSink (
  `topic` String,
  `partition` int,
  `headers` MAP<STRING, BYTES>,
  `leader-epoch` INT,
  `offset` bigint,
  `timestampType` string,
  `event_time` TIMESTAMP(3),
  `message` string --message表示读取kafka中存储的用户写入数据
) WITH (
  'connector' = 'kafka',
  'topic' = '***SinkKafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'format' = 'json'
);
insert into kafkaSink select * from kafkaSource;
```

4. 向Kafka的源表的topic中发送如下数据，Kafka topic为kafkaSource。

   具体操作可参考：**Kafka客户端接入示例**。

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```


版权所有 © 华为技术有限公司

5. 读取Kafka的结果表的topic，Kafka topic为kafkaSink。

   具体操作可参考：**Kafka客户端接入示例**。

```
{"topic":"kafkaSource","partition":1,"headers":{},"leader-epoch":0,"offset":4,"timestampType":"LogAppendTime","event_time":"2023-11-16 11:16:30.369","message":"{\"order_id\":\"202103251202020001\", \"order_channel\":\"miniAppShop\", \"order_time\":\"2021-03-25 12:02:02\", \"pay_amount\":\"60.00\", \"real_pay\":\"60.00\", \"pay_time\":\"2021-03-25 12:03:00\", \"user_id\":\"0002\", \"user_name\":\"Bob\", \"area_id\":\"330110\"}"}

{"topic":"kafkaSource","partition":0,"headers":{},"leader-epoch":0,"offset":6,"timestampType":"LogAppendTime","event_time":"2023-11-16 11:16:30.367","message":"{\"order_id\":\"202103241000000001\",\"order_channel\":\"webShop\",\"order_time\":\"2021-03-24 10:00:00\",\"pay_amount\":100.0,\"real_pay\":100.0,\"pay_time\":\"2021-03-24 10:02:03\",\"user_id\":\"0001\",\"user_name\":\"Alice\",\"area_id\":\"330106\"}"}

{"topic":"kafkaSource","partition":2,"headers":{},"leader-epoch":0,"offset":5,"timestampType":"LogAppendTime","event_time":"2023-11-16 11:16:30.368","message":"{\"order_id\":\"202103241606060001\",\"order_channel\":\"appShop\",\"order_time\":\"2021-03-24 16:06:06\",\"pay_amount\":200.0,\"real_pay\":180.0,\"pay_time\":\"2021-03-24 16:10:06\",\"user_id\":\"0001\",\"user_name\":\"Alice\",\"area_id\":\"330106\"}"}
```

示例 2：将 json 格式 DMS Kafka 作为源表，输出到 Kafka sink 中（适用于 Kafka 集群未开启 SASL_SSL 场景）

将Kafka作为源表，Kafka作为结果表，从Kafka中读取编码格式为json数据类型的数据，输出到日志文件中。

1. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，并提交运行。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource(
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaSourceTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE kafkaSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
```


版权所有 © 华为技术有限公司

```
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaSinkTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'format' = 'json'
);
insert into kafkaSink select * from kafkaSource;
```

4. 向Kafka的源表的topic中发送如下数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

5. 读取Kafka的结果表的topic，其数据结果参考如下：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

示例 3：将 DMS Kafka 作为源表，Print 作为结果表（适用于 Kafka 集群已开启 SASL_SSL 场景）

创建DMS的kafka集群，开启SASL_SSL，并下载SSL证书，将下载的证书client.jks上传到OBS桶中。

其中，properties.sasl.jaas.config字段包含账号密码，使用DEW进行加密。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1***:9093,***KafkaAddress2***:9093',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = '***obs://xx/client.jks***',  -- 用户上传证书的位置
  'properties.sasl.mechanism' = '***PLAIN***',
  'properties.security.protocol' = '***SASL_SSL***',
  'properties.sasl.jaas.config' = '***xx***',  -- dew凭据管理中的key，其值如：
org.apache.kafka.common.security.plain.PlainLoginModule required username=xx password=xx;
  'format' = 'json',
```

```
 'dew.endpoint' = 'kms.xx.com', --使用的DEW服务所在的endpoint信息
 'dew.csms.secretName' = 'xx', --DEW服务通用凭据的凭据名称
 'dew.csms.decrypt.fields' = 'properties.sasl.jaas.config', --其中properties.sasl.jaas.config字段值，需要利用
DEW凭证管理,进行解密替换
 'dew.csms.version' = 'v1'
);

CREATE TABLE ordersSink (
 order_id string,
 order_channel string,
 order_time timestamp(3),
 pay_amount double,
 real_pay double,
 pay_time string,
 user_id string,
 user_name string,
 area_id string
) WITH (
 'connector' = 'print'
);
 insert into ordersSink select * from ordersSource;
```

示例 4：将 Kafka（MRS 集群）作为源表，Print 作为结果表（适用于 Kafka 已开启 SASL_SSL 场景，MRS 使用 Kerberos 认证。）

- MRS集群请开启Kerberos认证。
- 在”组件管理 > Kafka > 服务配置”中查找配置项“ssl.mode.enable”，并设置为“true”，并重启kafka。
- 登录MRS集群的Manager，下载用户凭据：“系统设置 > 用户管理”，单击用户名后的“更多 > 下载认证凭据”。

  根据用户凭据生成相应的truststore.jks文件，并将用户凭据以及truststore.jks文件传入OBS中。

  具体方式请参考**客户端SSL加密功能使用说明**。
- 如果运行作业提示“Message stream modified (41)”，可能与JDK的版本有关系，可以尝试修改运行样例代码的JDK为8u_242以下版本或删除“krb5.conf”配置文件的“renew_lifetime = 0m”配置项。
- 端口请使用KafKa服务配置中设置的sasl_ssl.port端口，默认为21009。
- security.protocol请设置为SASL_SSL。
- with参数中properties.ssl.truststore.password字段使用dew进行加密。

```
CREATE TABLE ordersSource (
 order_id string,
 order_channel string,
 order_time timestamp(3),
 pay_amount double,
 real_pay double,
 pay_time string,
 user_id string,
 user_name string,
 area_id string
) WITH (
 'connector' = 'kafka',
 'topic' = 'kafkaTopic',
 'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
 'properties.group.id' = 'GroupId',
 'scan.startup.mode' = 'latest-offset',
 'properties.sasl.kerberos.service.name' = 'kafka', -- mrs集群中配置值
 'properties.connector.auth.open' = 'true',
 'properties.connector.kerberos.principal' = 'xx',  -- 用户名
 'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
 'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
```

 版权所有 © 华为技术有限公司

```
  'properties.security.protocol' = 'SASL_SSL',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',  -- dew凭据中的key
  'properties.sasl.mechanism' = 'GSSAPI',
  'format' = 'json',
  'dew.endpoint'='kms.xx.xx.com', --使用的DEW服务所在的endpoint信息
  'dew.csms.secretName'='xx', --DEW服务通用凭据的凭据名称
  'dew.csms.decrypt.fields'='properties.ssl.truststore.password', --其中properties.ssl.truststore.password字段
值，需要利用DEW凭证管理,进行解密替换
  'dew.csms.version'='v1'
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);
 insert into ordersSink select * from ordersSource;
```

示例 5：将 Kafka（MRS 集群）作为源表，Print 作为结果表（适用于 Kafka 已开启 SASL_SSL 场景，MRS 使用 SASL_PAINTEXT 的 Kerberos 认证。）

- MRS集群请开启Kerberos认证。
- 将“组件管理 > Kafka > 服务配置”中查找配置项“ssl.mode.enable”，并设置为“True”， 并重启kafka。
- 登录MRS集群的Manager，下载用户凭据“系统设置 > 用户管理”，单击用户名后的“更多 > 下载认证凭据”，并上传到OBS中。
- 如果运行提示“Message stream modified (41)”的错误，可能与JDK的版本有关系，可以尝试修改运行样例代码的JDK为8u_242以下版本或删除“krb5.conf”配置文件的“renew_lifetime = 0m”配置项。
- 端口请使用KafKa服务配置中设置的sasl.port端口，默认为21007。
- security.protocol请设置为SASL_PLAINTEXT。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'properties.sasl.kerberos.service.name' = 'kafka', -- mrs集群中配置
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = '***xx***',
  'properties.connector.kerberos.krb5' = '***obs://xx/krb5.conf***',
  'properties.connector.kerberos.keytab' = '***obs://xx/user.keytab***',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
```

```
  'format' = 'json'
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);
 insert into ordersSink select * from ordersSource;
```

示例 6：将 Kafka（MRS 集群）作为源表，Print 作为结果表（适用于 Kafka 已开启 SSL 场景，MRS 未开启 Kerberos 认证。）

- MRS集群请不要开启Kerberos认证。
- 登录MRS集群的Manager，下载用户凭据：“系统设置 > 用户管理”。单击用户名后的“更多 > 下载认证凭据”。

  根据用户凭据生成相应的truststore.jks文件，并将用户凭据以及truststore.jks文件传入OBS中。

  具体方式请参考**客户端SSL加密功能使用说明**。
- 端口请注意使用KafKa服务配置中设置的ssl.port端口，默认值为9093。
- with参数中security.protocol请设置为SSL。
- MRS集群kafka服务配置中，设置ssl.mode.enable请设置为true，并重启kafka
- with参数中properties.ssl.truststore.password字段使用dew进行加密。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = '***obs://xx/truststore.jks***',
  'properties.ssl.truststore.password' = '***xx***',  -- dew凭据管理的key，其值为生成truststore.jks时设置的密码
  'properties.security.protocol' = '***SSL***',
  'format' = 'json',
  'dew.endpoint' = '***kms.xx.com***', --使用的DEW服务所在的endpoint信息
  'dew.csms.secretName' = '***xx***', --DEW服务通用凭据的凭据名称
  'dew.csms.decrypt.fields' = '***properties.ssl.truststore.password***', --其中
properties.ssl.truststore.password字段值，需要利用DEW凭证管理,进行解密替换
  'dew.csms.version' = 'v1'
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
```

 版权所有 © 华为技术有限公司

```
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);
 insert into ordersSink select * from ordersSource;
```

常见问题

- **Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?**

  ```
  org.apache.kafka.common.errors.TimeoutException: Timeout expired while fetching topic metadata
  ```

  跨源未绑定或未绑定成功，或是Kafka集群安全组未配置放通DLI队列的网段地址。重新配置跨源，或者Kafka集群安全组放通DLI队列的网段地址。

  具体操作请参考**增强型跨源连接**。

- **Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?**

  ```
  Caused by: java.lang.RuntimeException: RealLine:45;Table 'default_catalog.default_database.printSink' declares persistable metadata columns, but the underlying DynamicTableSink doesn't implement the SupportsWritingMetadata interface. If the column should not be persisted, it can be declared with the VIRTUAL keyword.
  ```

  sink表中定义了metadata类型，但是Print connector并不支持把sink表中的matadata去掉即可。

2.5.14 MySql CDC

功能描述

MySQL的CDC源表，即MySQL的流式源表，会先读取数据库的历史全量数据，并平滑切换到Binlog读取上，保证数据的完整读取。

表 2-59 支持类别

类别	详情
支持表类型	源表

前提条件

- MySQL CDC要求MySQL版本为5.6，5.7或8.0.x。
- with参数中字段只能使用单引号，不能使用双引号。
- 该场景作业需要DLI与MySQL建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。
- MySQL已开启了Binlog，并且binlog_row_image设置为FULL。
- 已创建MySQL用户，并授予了SELECT、 SHOW DATABASES 、REPLICATION SLAVE和REPLICATION CLIENT权限。注意: 在

 版权所有 © 华为技术有限公司

scan.incremental.snapshot.enabled 参数已启用时（默认情况下已启用）时，不再需要授予 reload 权限。

```
GRANT SELECT, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'user' IDENTIFIED BY 'password';
```

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 为每个Reader设置不同的Server ID
  - 每个用于读取Binlog的MySQL客户端都应该有一个唯一的Server ID，确保MySQL服务器能够区分不同的客户端并维护各自的Binlog读取位置。
  - 如果不同的作业共享相同的Server ID，可能会导致从错误的Binlog位置读取数据，从而引发数据不一致的问题。
  - 可以通过SQL Hints为每个Source Reader分配唯一的Server ID，例如使用 SELECT * FROM source_table /*+ OPTIONS('server-id'='5401-5404') */ ; 为4个Source Readers分配唯一的 Server ID。
- 设置MySQL会话超时时间

  当为大型数据库创建初始一致快照时，您建立的连接可能会在读取表时碰到超时问题。您可以通过在MySQL侧配置 interactive_timeout 和 wait_timeout来解决此类问题。
  - interactive_timeout: 服务器在关闭交互连接之前等待活动的秒数。 更多信息请参考 **MySQL Documentations**.
  - wait_timeout: 服务器在关闭非交互连接之前等待活动的秒数。 更多信息请参考 **MySQL Documentations**.
- 使用无主键表时的注意事项：
  - 使用无主键表必须设置 scan.incremental.snapshot.chunk.key-column，且只能选择非空类型的一个字段。
  - 配置 scan.incremental.snapshot.chunk.key-column 时，如果表中存在索引，请使用索引中的列来加快 select 度。

    无主键表的处理语义由 scan.incremental.snapshot.chunk.key-column 指定的列的行为决定：
    - 如果指定的列不存在更新操作，此时可以保证 Exactly once 语义。
    - 如果指定的列存在更新操作，此时只能保证 At least once 语义。但可以结合下游，通过指定下游主键，结合幂等性操作来保证数据的正确性。
- MySQL CDC源表暂不支持定义Watermark。如果您需要进行窗口聚合，请参考**常见问题**描述。
- 如果连接DWS、MySQL等支持upsert的sink源，需要在sink表的创建语句中定义主键，请参考**示例1：利用MySQL-CDC实时读取RDS MySQL中的数据及其元数据**中printSink建表语句。

版权所有 © 华为技术有限公司

支持特性

- 增量快照读取

  增量快照读取是一种读取表快照的新机制。与旧的快照机制相比，增量快照具有许多优点，包括：

  - 在快照读取期间，Source 支持并发读取，
  - 在快照读取期间，Source 支持进行 chunk 粒度的 checkpoint，
  - 在快照读取之前，Source 不需要数据库锁权限。

  如果希望 source 并行运行，则每个并行 reader 都应该具有唯一的 server id，因此server id的范围必须类似于 5400-6400， 且范围必须大于并行度。在增量快照读取过程中，MySQL CDC Source 首先通过表的主键将表划分成多个块（chunk）， 然后 MySQL CDC Source 将多个块分配给多个 reader 以并行读取表的数据。

- 无锁算法

  MySQL CDC source 使用 增量快照算法, 避免了数据库锁的使用，因此不需要“RELOAD” 权限。

- 并发读取

  增量快照读取提供了并行读取快照数据的能力。

- 全量阶段支持checkpoint

  增量快照读取提供了在区块级别执行检查点的能力。它使用新的快照读取机制解决了以前版本中的检查点超时问题。

语法格式

```
create table mySqlCdcSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'mysql-cdc',
  'hostname' = 'mysqlHostname',
  'username' = 'mysqlUsername',
  'password' = 'mysqlPassword',
  'database-name' = 'mysqlDatabaseName',
  'table-name' = 'mysqlTableName'
);
```

参数说明

表 2-60 源表参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'mysql-cdc'。
hostname	是	无	String	MySQL 数据库服务器的 IP 地址或主机名。
username	是	无	String	连接到 MySQL 数据库服务器时要使用的 MySQL 用户的名称。

 版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
password	是	无	String	连接 MySQL 数据库服务器时使用的密码。
database-name	是	无	String	要监视的 MySQL 服务器的数据库名称。 数据库名称还支持正则表达式，以监视多个与正则表达式匹配的表。 ● 前缀匹配：^(test).* 匹配前缀为test的数据库名，例如test1、test2等。 ● 后缀匹配：.*[p$] 匹配后缀为p的数据库名，例如cdcp、edcp等。 ● 特定匹配：txc 匹配具体的数据库名。
table-name	是	无	String	需要监视的 MySQL 数据库的表名。表名支持正则表达式，以监视满足正则表达式的多个表。 **说明** MySQL CDC 连接器在正则匹配表名时，会把用户填写的 database-name，table-name 通过字符串 `\\.` 连接成一个全路径的正则表达式，然后使用该正则表达式和 MySQL 数据库中表的全限定名进行正则匹配。 ● 前缀匹配：^(test).* 匹配前缀为test的表名，例如test1、test2等。 ● 后缀匹配：.*[p$] 匹配后缀为p的表名，例如cdcp、edcp等。 ● 特定匹配：txc 匹配具体的表名。
port	否	3306	Integer	MySQL 数据库服务器的整数端口号。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
server-id	否	无	String	读取数据使用的 server id，server id 可以是个整数或者一个整数范围，比如 '5400' 或 '5400-5408'。 建议在 'scan.incremental.snapshot.enabled' 参数为启用时，配置成整数范围。 在当前 MySQL 集群中运行的所有 slave 节点，标记每个 salve 节点的 id 都必须是唯一的。 所以当连接器加入 MySQL 集群作为另一个 slave 节点（并且具有唯一 id 的情况下），它就可以读取 binlog。 默认情况下，连接器会在 5400 和 6400 之间生成一个随机数，但是我们建议用户明确指定 Server id。
scan.incremental.snapshot.enabled	否	true	Boolean	增量快照是一种读取表快照的新机制，与旧的快照机制相比， 增量快照有许多优点，包括： ● 在快照读取期间，Source 支持并发读取 ● 在快照读取期间，Source 支持进行 chunk 粒度的 checkpoint ● 在快照读取之前，Source 不需要数据库锁权限。 如果希望 Source 并行运行，则每个并行 Readers 都应该具有唯一的 Server id，所以 Server id 必须是类似 `5400-6400` 的范围，并且该范围必须大于并行度。
scan.incremental.snapshot.chunk.size	否	8096	Integer	表快照的块大小（行数），读取表的快照时，捕获的表被拆分为多个块。
scan.snapshot.fetch.size	否	1024	Integer	读取表快照时每次读取数据的最大条数。
scan.incremental.snapshot.chunk.key-column	否	无	String	无主键表必填。 可以指定某一列作为快照阶段切分分片的切分列，只能选择非空类型的一个字段。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
scan.startup.mode	否	initial	String	MySQL CDC 消费者可选的启动模式， 合法的模式为 "initial"，"earliest-offset"，"latest-offset"，"specific-offset" 和 "timestamp"。 ● initial（默认）：在第一次启动时对受监视的数据库表执行初始快照，并继续读取最新的 binlog。 ● earliest-offset：跳过快照阶段，从可读取的最早 binlog 位点开始读取。 ● latest-offset：首次启动时，从不对受监视的数据库表执行快照，连接器仅从 binlog 的结尾处开始读取，这意味着连接器只能读取在连接器启动之后的数据更改。 ● specific-offset：跳过快照阶段，从指定的 binlog 位点开始读取。位点可通过 binlog 文件名和位置指定，或者在 GTID 在集群上启用时通过 GTID 集合指定。 ● timestamp：跳过快照阶段，从指定的时间戳开始读取 binlog 事件。
scan.startup.specific-offset.file	否	无	String	在 "specific-offset" 启动模式下，启动位点的 binlog 文件名。
scan.startup.specific-offset.pos	否	无	Long	在 "specific-offset" 启动模式下，启动位点的 binlog 文件位置。
scan.startup.specific-offset.gtid-set	否	无	String	在 "specific-offset" 启动模式下，启动位点的 GTID 集合。
scan.startup.specific-offset.skip-events	否	无	Long	在指定的启动位点后需要跳过的事件数量。
scan.startup.specific-offset.skip-rows	否	无	Long	在指定的启动位点后需要跳过的数据行数量。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
server-time-zone	否	无	String	数据库服务器中的会话时区。 例如： "Asia/Shanghai". 它控制 MYSQL 中的时间戳类型如何转换为字符串。如果没有设置，则使用ZoneId.systemDefault()来确定服务器时区。
debezium.min.row.count.to.stream.result	否	1000	Integer	在快照操作期间，连接器将查询每个包含的表，以生成该表中所有行的读取事件。 此参数确定 MySQL 连接是否将表的所有结果拉入内存（速度很快，但需要大量内存）， 或者结果是否需要流式传输（传输速度可能较慢，但适用于非常大的表）。 该值指定了在连接器对结果进行流式处理之前，表必须包含的最小行数，默认值为1000。 将此参数设置为`0`以跳过所有表大小检查，并始终在快照期间对所有结果进行流式处理。
connect.timeout	否	30s	Duration	连接器在尝试连接到 MySQL 数据库服务器后超时前应等待的最长时间。
connect.max-retries	否	3	Integer	连接器应重试以建立 MySQL 数据库服务器连接的最大重试次数。
connection.pool.size	否	20	Integer	连接池大小。
jdbc.properties.*	否	无	String	传递自定义 JDBC URL 属性的选项。 用户可以传递自定义属性，如 'jdbc.properties.useSSL' = 'false'.
heartbeat.interval	否	30s	Duration	用于跟踪最新可用 binlog 偏移的发送心跳事件的间隔。
debezium.*	否	无	String	将 Debezium 的属性传递给 Debezium 嵌入式引擎，该引擎用于从 MySQL 服务器捕获数据更改。 例如: 'debezium.snapshot.mode' = 'never'. 查看更多关于 **Debezium 的 MySQL 连接器属性**。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
scan.incremental.close-idle-reader.enabled	否	false	Boolean	是否在快照结束后关闭空闲的 Reader。 此特性需要 'execution.checkpointing.checkpoints-after-tasks-finish.enabled' 需要设置为 true。

元数据

元数据可以在 DDL 中作为只读（虚拟）meta 列声明。

表 2-61 元数据

Key	数据类型	说明
table_name	STRING NOT NULL	当前记录所属的表名称。
database_name	STRING NOT NULL	当前记录所属的库名称。
op_ts	TIMESTAMP_LTZ(3) NOT NULL	当前记录表在数据库中更新的时间。 如果从表的快照而不是 binlog 读取记录，该值将始终为0。

数据类型映射

表 2-62 数据类型映射

MySQL类型	Flink SQL类型	备注
TINYINT	TINYINT	-
SMALLINT TINYINT UNSIGNED TINYINT UNSIGNED ZEROFILL	SMALLINT	-
INT MEDIUMINT SMALLINT UNSIGNED SMALLINT UNSIGNED ZEROFILL	INT	-

版权所有 © 华为技术有限公司

MySQL类型	Flink SQL类型	备注
BIGINT INT UNSIGNED INT UNSIGNED ZEROFILL MEDIUMINT UNSIGNED MEDIUMINT UNSIGNED ZEROFILL	BIGINT	-
BIGINT UNSIGNED BIGINT UNSIGNED ZEROFILL SERIAL	DECIMAL(20, 0)	-
FLOAT FLOAT UNSIGNED FLOAT UNSIGNED ZEROFILL	FLOAT	-
REAL REAL UNSIGNED REAL UNSIGNED ZEROFILL DOUBLE DOUBLE UNSIGNED DOUBLE UNSIGNED ZEROFILL DOUBLE PRECISION DOUBLE PRECISION UNSIGNED DOUBLE PRECISION UNSIGNED ZEROFILL	DOUBLE	-
NUMERIC(p, s) NUMERIC(p, s) UNSIGNED NUMERIC(p, s) UNSIGNED ZEROFILL DECIMAL(p, s) DECIMAL(p, s) UNSIGNE DDECIMAL(p, s) UNSIGNED ZEROFILL FIXED(p, s) FIXED(p, s) UNSIGNED FIXED(p, s) UNSIGNED ZEROFILL where p <= 38	DECIMAL(p, s)	-

版权所有 © 华为技术有限公司

MySQL类型	Flink SQL类型	备注
NUMERIC(p, s) NUMERIC(p, s) UNSIGNED NUMERIC(p, s) UNSIGNED ZEROFILL DECIMAL(p, s) DECIMAL(p, s) UNSIGNED DECIMAL(p, s) UNSIGNED ZEROFILL FIXED(p, s) FIXED(p, s) UNSIGNED FIXED(p, s) UNSIGNED ZEROFILL where 38 < p <= 65	STRING	在 MySQL 中，十进制数据类型的精度高达 65，但在 Flink 中，十进制数据类型的精度仅限于 38。 如果定义精度大于 38 的十进制列，则应将其映射到字符串以避免精度损失。
BOOLEAN TINYINT(1) BIT(1)	BOOLEAN	-
DATE	DATE	-
TIME [(p)]	TIME [(p)]	-
TIMESTAMP [(p)] DATETIME [(p)]	TIMESTAMP [(p)]	-
CHAR(n)	CHAR(n)	-
VARCHAR(n)	VARCHAR(n)	-
BIT(n)	BINARY(⌈n/8⌉)	-
BINARY(n)	BINARY(n)	-
VARBINARY(N)	VARBINARY(N)	-
TINYTEXT TEXT MEDIUMTEXT LONGTEXT	STRING	-
TINYBLOB BLOB MEDIUMBLOB LONGBLOB	BYTES	目前，对于 MySQL 中的 BLOB 数据类型，仅支持长度不大于 2147483647（2**31-1）的 blob。
YEAR	INT	-
ENUM	STRING	-

版权所有 © 华为技术有限公司

MySQL类型	Flink SQL类型	备注
JSON	STRING	JSON 数据类型将在 Flink 中转换为 JSON 格式的字符串。
SET	ARRAY<STRING>	因为MySQL中的SET数据类型是一个字符串对象，可以有零个或多个值 它应该始终映射到字符串数组。
GEOMETRY POINT LINESTRING POLYGON MULTIPOINT MULTILINESTRING MULTIPOLYGON GEOMETRYCOLLECTION	STRING	MySQL中的空间数据类型将转换为具有固定Json格式的字符串。

示例 1：利用 MySQL-CDC 实时读取 RDS MySQL 中的数据及其元数据

该示例是利用MySQL-CDC实时读取RDS MySQL中的数据及其元数据，并写入到Print结果表中。

本示例使用RDS MySQL数据库引擎版本为MySQL 5.7.33。

1. 参考**增强型跨源连接**，根据MySQL所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置MySQL的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据MySQL的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 在MySQL中创建用户test，并授权，SQL语句参考如下：

```
CREATE USER 'test'@'%' IDENTIFIED BY 'xxx';
GRANT SELECT, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'test';
FLUSH PRIVILEGES;
```

4. 在MySQL中的flink数据库下创建相应的表，表名为cdc_order，SQL语句参考如下(该语句需要用户拥有CREATE权限)：

```
CREATE TABLE `flink`.`cdc_order` (
  `order_id` VARCHAR(32) NOT NULL,
  `order_channel` VARCHAR(32) NULL,
  `order_time` VARCHAR(32) NULL,
  `pay_amount` DOUBLE  NULL,
  `real_pay` DOUBLE  NULL,
  `pay_time` VARCHAR(32) NULL,
  `user_id` VARCHAR(32) NULL,
  `user_name` VARCHAR(32) NULL,
  `area_id` VARCHAR(32) NULL,
  PRIMARY KEY (`order_id`)
)  ENGINE = InnoDB
  DEFAULT CHARACTER SET = utf8mb4
  COLLATE = utf8mb4_general_ci;
```


版权所有 © 华为技术有限公司

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
create table mysqlCdcSource(
  database_name STRING METADATA VIRTUAL,
  table_name STRING METADATA VIRTUAL,
  operation_ts TIMESTAMP_LTZ(3) METADATA FROM 'op_ts' VIRTUAL,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id STRING,
  primary key(order_id) not enforced
) with (
  'connector' = 'mysql-cdc',
  'hostname' = '***mysqlHostname***',
  'username' = '***mysqlUsername***',
  'password' = '***mysqlPassword***',
  'database-name' = '***mysqlDatabaseName***',
  'table-name' = '***mysqlTableName***'
);

create table printSink(
  database_name string,
  table_name string,
  operation_ts TIMESTAMP_LTZ(3),
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id STRING,
  primary key(order_id) not enforced
) with (
  'connector' = 'print'
);
insert into printSink select * from mysqlCdcSource;
```

6. 在MySQL中执行以下命令插入测试数据(该语句需要用户有相应的权限)。

```
insert into flink.cdc_order values
('202103241000000001','webShop','2021-03-24 10:00:00','100.00','100.00','2021-03-24
10:02:03','0001','Alice','330106'),
('202103241606060001','appShop','2021-03-24 16:06:06','200.00','180.00','2021-03-24
16:10:06','0001','Alice','330106');

delete from flink.cdc_order  where order_channel = 'webShop';
insert into flink.cdc_order values('202103251202020001','miniAppShop','2021-03-25
12:02:02','60.00','60.00','2021-03-25 12:03:00','0002','Bob','330110');
```

7. 按照如下方式查看taskmanager.out文件中的数据结果：

   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

 版权所有 © 华为技术有限公司

```
 +I[flink, cdc_order, 2023-11-10T07:41:12Z, 202103241000000001, webShop, 2021-03-24 10:00:00, 100.0, 100.0, 2021-03-24 10:02:03, 0001, Alice, 330106]
+I[flink, cdc_order, 2023-11-10T07:41:12Z, 202103241606060001, appShop, 2021-03-24 16:06:06, 200.0, 180.0, 2021-03-24 16:10:06, 0001, Alice, 330106]
-D[flink, cdc_order, 2023-11-10T07:41:59Z, 202103241000000001, webShop, 2021-03-24 10:00:00, 100.0, 100.0, 2021-03-24 10:02:03, 0001, Alice, 330106]
+I[flink, cdc_order, 2023-11-10T07:42:00Z, 202103251202020001, miniAppShop, 2021-03-25 12:02:02, 60.0, 60.0, 2021-03-25 12:03:00, 0002, Bob, 330110]
```

示例 2：无主键表启用增量快照功能并指定快照的切分列

假设我们有一个名为orders的表，它没有主键，并且我们想使用Flink CDC从这个表中读取数据。

在这个例子中，我们为orders表启用了增量快照功能，并指定了order_id和product_name两列作为分块的关键字列。这样，即使orders表没有主键，Flink CDC仍然能够根据这些列来进行增量快照读取，从而有效地同步数据。

```
CREATE TABLE orders (
    order_id BIGINT,
    product_name STRING,
    quantity INT,
    order_time TIMESTAMP(3),
    -- 其他字段...
    watermark for order_time as order_time -- 如果需要watermark
) WITH (
    'connector' = 'mysql-cdc',
    'hostname' = 'your.mysql.host',
    'port' = '3306',
    'username' = 'your_username',
    'password' = 'your_password',
    'database-name' = 'your_database',
    'table-name' = 'orders',
    'scan.incremental.snapshot.enabled' = 'true',
    'scan.incremental.snapshot.chunk.size' = '1000',
    'scan.incremental.snapshot.chunk.key-column' = 'order_id,product_name'
);
```

常见问题

Q：MySQL CDC源表不支持定义Watermark，怎么进行窗口聚合?

A：可以采用非窗口聚合的方式，即将时间字段转换成窗口值，然后根据窗口值进行GROUP BY聚合。

例如：基于上述示例，统计每分钟的订单数，脚本如下（其中order_time为string类型，表示订单的时间）。

```
insert into printSink select DATE_FORMAT(order_time, 'yyyy-MM-dd HH:mm'), count(*) from mysqlCdcSource group by DATE_FORMAT(order_time, 'yyyy-MM-dd HH:mm');
```

2.5.15 Print

功能描述

Print connector用于将用户输出的数据打印到taskmanager中的error文件或者out文件中，方便用户查看，主要用于代码调试，查看输出结果。

前提条件

无。

 版权所有 © 华为技术有限公司

注意事项

- Print结果表支持以下四种格式内容输出：

打印内容	条件1	条件2
标识符:任务 ID> 输出数据	需要提供前缀打印标识符，即创建Print表时在with参数中指定print-identifier。	parallelism > 1
标识符> 输出数据	需要提供前缀打印标识符，即创建Print表时在with参数中指定print-identifier。	parallelism == 1
任务 ID> 输出数据	不需要提供前缀打印标识符，即创建Print表时在with参数中不指定print-identifier。	parallelism > 1
输出数据	不需要提供前缀打印标识符，即创建Print表时在with参数中不指定print-identifier。	parallelism == 1

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table printSink (
  attr_name attr_type
  (',' attr_name attr_type) *
  (',' PRIMARY KEY (attr_name,...) NOT ENFORCED)
) with (
  'connector' = 'print',
  'print-identifier' = '',
  'standard-error' = ''
);
```

参数说明

表 2-63 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	固定为：print。
print-identifier	否	无	String	配置一个标识符作为输出数据的前缀。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
standard-error	否	false	Boolean	该值只能为true或false，默认为false。 • 如果为true，则表示输出数据到taskmanager的error文件中。 • 如果为false，则表示输出数据到taskmanager的out中。
sink.parallelism	否	无	Integer	为Print结果表定义并行度。默认情况下，并行度由框架决定，与上游并行度一致。

示例

参考**创建Flink OpenSource作业**，创建flink opensource sql作业，运行如下作业脚本，通过DataGen表产生随机数据并输出到Print结果表中。

```
create table dataGenSource(
  user_id string,
  amount int
) with (
  'connector' = 'datagen',
  'rows-per-second' = '1', --每秒生成一条数据
  'fields.user_id.kind' = 'random', --为字段user_id指定random生成器
  'fields.user_id.length' = '3' --限制user_id长度为3
);

create table printSink(
  user_id string,
  amount int
) with (
  'connector' = 'print',
  'print-identifier' = '', --配置数据前缀
  'standard-error' = 'false', --输出数据到taskmanager的out文件中
  'sink.parallelism' = '2' --配置并行度
);

insert into printSink select * from dataGenSource;
```

该作业提交后，作业状态变成“运行中”，后续您可通过如下操作查看输出结果。

- 方法一：
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
  c. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
- 方法二：如果在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
  c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

版权所有 © 华为技术有限公司

- 方法三：如果是新版本队列，可以通过如下操作查看。
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 单击对应的Flink作业名称，选择“日志列表”。
  c. 在左上角下拉框选择对应的taskmanager名称，单击taskmanager.out文件查看结果日志。

2.5.16 Redis

2.5.16.1 Redis 源表

功能描述

创建source流从Redis获取数据，作为作业的输入数据。

前提条件

创建该作业前，需要建立DLI和Redis的增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 如果需要获取key的值，则可以通过在Flink中设置主键获取，主键字段即对应Redis的key。
- 如果定义主键，则不能够定义复合主键，即主键只能是一个字段，不能是多个字段。
- schema-syntax取值约束：
  - 当schema-syntax为map或array时，非主键字段最多只能有一个，且需要为相应的map或array类型。
  - 当schema-syntax为fields-scores时，非主键字段个数需要为偶数，且除主键字段外，每两个字段的第二个字段的数据类型需要为double，该字段的值视为前一个字段的score。其示例如下：

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  score1 double,
  order_channel string,
  score2 double,
  order_time string,
  score3 double,
  pay_amount double,
  score4 double,
```


版权所有 © 华为技术有限公司

```
  real_pay double,
  score5 double,
  pay_time string,
  score6 double,
  user_id string,
  score7 double,
  user_name string,
  score8 double,
  area_id string,
  score9 double,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields-scores'
);
```

- data-type取值约束：
  - 当data-type为set时，Flink中定义的非主键字段的数据类型必须相同。
  - 当data-type为sorted-set并且schema-syntax为fields和array时，只能读取redis的sorted set中的值，而不能读取score。
  - 当data-type为string时，只能有一个非主键字段。
  - 当data-type为sorted-set，且schema-syntax为map时，除主键字段外，只能有一个非主键字段。

    该非主键字段需要为map类型，同时该字段map的value需要为double类型，表示score，该字段的map的key表示redis的set中的值。
  - 当data-type为sorted-set，且schema-syntax为array-scores时，除主键字段外，只能有两个非主键字段，且这两个字段的类型需要为array。

    两个字段其中第一个字段类型是array，表示Redis的set中的值；第二个字段类型为array<double>，表示相应索引的score。其示例如下：

```
CREATE TABLE redisSink (
  order_id string,
  arrayField Array<String>,
  arrayScore array<double>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
  "default-score" = '3',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'array-scores'
);
```

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
  ,PRIMARY KEY (attr_name, ...) NOT ENFORCED
)
with (
  'connector' = 'redis',
  'host' = ''
);
```

 版权所有 © 华为技术有限公司

参数说明

表 2-64 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'redis'。
host	是	无	String	redis连接地址。
port	否	6379	Integer	redis连接端口。
password	否	无	String	redis认证密码。
namespace	否	无	String	redis key的namespace
delimiter	否	:	String	redis的key和namespace之间的分隔符。
data-type	否	hash	String	redis的数据类型，有下列选项： • hash • list • set • sorted-set • string data-type取值约束详见**data-type取值约束**说明。
schema-syntax	否	fields	String	redis的schema语义，包含以下值（其具体使用请参考**注意事项**和**常见问题**）： • fields：适用于所有数据类型 • fields-scores：适用于sorted set数据类型 • array：适用于list、set、sorted set数据类型 • array-scores：适用于sorted set数据类型 • map：适用于hash、sorted set数据类型 schema-syntax取值约束详见**schema-syntax取值约束**说明。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
deploy-mode	否	standalone	String	Redis集群的部署模式，支持standalone、master-replica、cluster。默认为standalone。 Redis实例类型不同配置的部署模式不同： 单机、主备、proxy集群实例都选择standalone， cluster实例选择cluster。
retry-count	否	5	Integer	连接redis集群的尝试次数。
connection-timeout-millis	否	10000	Integer	尝试连接redis集群时的最大超时时间。
commands-timeout-millis	否	2000	Integer	等待操作完成响应的最大时间。
rebalancing-timeout-millis	否	15000	Integer	redis集群失败时的休眠时间。
scan-keys-count	否	1000	Integer	每次扫描时读取的数量。
default-score	否	0	Double	当data-type设置为“sorted-set”时的默认score。
deserialize-error-policy	否	fail-job	Enum	数据解析失败时的处理方式。枚举类型，包含以下值： ● fail-job：作业失败 ● skip-row：跳过当前数据 ● null-field：设置当前数据为null
skip-null-values	否	true	Boolean	是否跳过null。
ignore-retractions	否	false	Boolean	连接器应忽略更新插入/撤回流模式下的收回消息。
key-column	否	无	String	Redis 表schema的key
source.parallelism	否	无	int	定义源的自定义并行度。默认情况下，如果未定义此选项，使用全局配置来的并行度。

示例

该示例是从DCS Redis数据源中读取数据，并写入Print到结果表中，其具体步骤如下：

版权所有 © 华为技术有限公司

1. 参考**增强型跨源连接**，根据redis所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Redis的安全组，添加入向规则使其对Flink的队列网段放通。

   参考**测试地址连通性**根据redis的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 在Redis客户端中执行如下命令，向不同的key中插入数据，以hash形式存储：

```
HMSET redisSource order_id 202103241000000001 order_channel webShop order_time "2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24 10:02:03" user_id 0001 user_name Alice area_id 330106

HMSET redisSource1 order_id 202103241606060001 order_channel appShop order_time "2021-03-24 16:06:06" pay_amount 200.00 real_pay 180.00 pay_time "2021-03-24 16:10:06" user_id 0001 user_name Alice area_id 330106

HMSET redisSource2 order_id 202103251202020001 order_channel miniAppShop order_time "2021-03-25 12:02:02" pay_amount 60.00 real_pay 60.00 pay_time "2021-03-25 12:03:00" user_id 0002 user_name Bob area_id 330110
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本读取Redis中hash格式的数据。

   如下脚本中的加粗参数请根据实际环境修改。

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  primary key (redisKey) not enforced  --获取redis中key的值
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'hash',
  'deploy-mode' = '***master-replica***'
);

CREATE TABLE printSink (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from redisSource;
```

5. 按照如下方式查看taskmanager.out文件中的数据结果：
   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

版权所有 © 华为技术有限公司

数据结果参考如下：

```
+I(redisSource1,202103241606060001,appShop,2021-03-24 16:06:06,200.0,180.0,2021-03-24 16:10:06,0001,Alice,330106)
+I(redisSource,202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
+I(redisSource2,202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25 12:03:00,0002,Bob,330110)
```

常见问题

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决？

  ```
  Caused by: org.apache.flink.client.program.ProgramInvocationException: The main method caused an error: RealLine:36;Usage of 'set' data-type and 'fields' schema syntax in source Redis connector with multiple non-key column types. As 'set' in Redis is not sorted, it's not possible to map 'set's values to table schema with different types.
  ```

  A：data-type为set类型时，flink中非主键字段的数据类型不相同，导致如上报错。data-type为set类型时，Flink中定义的非主键字段的数据类型必须相同。

- Q：当使用data-type为hash时，那么schema-syntax为fields和map有什么区别？

  A：当schema-syntax为fields时，会将Redis的key中hash值赋给flink中同名相应字段；当schema-syntax为map时，会将Redis的每个hash中的hashkey和hashvalue放入一个map中，该map即为flink中相应字段的值，即这个map中包含Redis中某个key的所有hashkey和hashvalue。

  - 对于fields而言：

    i. 向Redis中插入如下数据

    ```
    HMSET redisSource order_id 202103241000000001 order_channel webShop order_time "2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24 10:02:03" user_id 0001 user_name Alice area_id 330106
    ```

    ii. 当使用schema-syntax为fields时，作业脚本参考如下：

    ```
    CREATE TABLE redisSource (
      redisKey string,
      order_id string,
      order_channel string,
      order_time string,
      pay_amount double,
      real_pay double,
      pay_time string,
      user_id string,
      user_name string,
      area_id string,
      primary key (redisKey) not enforced
    ) WITH (
      'connector' = 'redis',
      'host' = '***RedisIP***',
      'password' = '***RedisPassword***',
      'data-type' = 'hash',
      'deploy-mode' = '***master-replica***'
    );

    CREATE TABLE printSink (
      redisKey string,
      order_id string,
      order_channel string,
      order_time string,
      pay_amount double,
      real_pay double,
      pay_time string,
      user_id string,
      user_name string,
      area_id string
    ) WITH (
      'connector' = 'print'
    );
    ```

 版权所有 © 华为技术有限公司

```
insert into printSink select * from redisSource;
```

iii. 作业运行结果如下：

```
+I(redisSource,202103241000000001,webShop,2021-03-24
10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
```

- 对于map而言：

i. 向Redis中插入如下数据：

```
HMSET redisSource order_id 202103241000000001 order_channel webShop order_time
"2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24
10:02:03" user_id 0001 user_name Alice area_id 330106
```

ii. 当使用schema-syntax为map时，其作业脚本参考如下：

```
CREATE TABLE redisSource (
  redisKey string,
  order_result map<string, string>,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'hash',
  'deploy-mode' = '***master-replica***',
  'schema-syntax' = 'map'
);

CREATE TABLE printSink (
  redisKey string,
  order_result map<string, string>
) WITH (
  'connector' = 'print'
);

insert into printSink select * from redisSource;
```

iii. 作业运行结果如下：

```
+I(redisSource,{user_id=0001, user_name=Alice, pay_amount=100.00, real_pay=100.00,
order_time=2021-03-24 10:00:00, area_id=330106, order_id=202103241000000001,
order_channel=webShop, pay_time=2021-03-24 10:02:03})
```

2.5.16.2 Redis 结果表

功能描述

DLI将Flink作业的输出数据输出到Redis中。Redis是一种支持Key-Value等多种数据结构的存储系统。可用于缓存、事件发布或订阅、高速队列等场景，提供字符串、哈希、列表、队列、集合结构直接存取，基于内存，可持久化。有关Redis的详细信息，请访问Redis官方网站**https://redis.io/**。

前提条件

DLI要建立与Redis的增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

版权所有 © 华为技术有限公司

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 如果未在创建Redis结果表的语句中定义Redis key的字段，则会使用生成的uuid作为key。
- 如果需要指定Redis中的key，则需要在flink的Redis结果表中定义主键，该主键的值即为key。
- Redis结果表如果定义主键，则不能够定义复合主键，即主键只能是一个字段，不能是多个字段。
- schema-syntax取值约束：
  - 当schema-syntax为map或array时，非主键字段最多只能只有一个，且需要为相应的map或array类型。
  - 当schema-syntax为fields-scores时，非主键字段个数需要为偶数，且除主键字段外，每两个字段的第二个字段的类型需要为double，会将该字段的值视为前一个字段的score。其示例如下：

```
CREATE TABLE redisSink (
  order_id string,
  order_channel string,
  order_time double,
  pay_amount STRING,
  real_pay double,
  pay_time string,
  user_id double,
  user_name string,
  area_id double,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'sorted-set',
  'deploy-mode' = '***master-replica***',
  'schema-syntax' = 'fields-scores'
);
```

- data-type取值约束：
  - 当data-type为string时，只能有一个非主键字段。
  - 当data-type为sorted-set，且schema-syntax为fields和array时，会使用default-score作为score。
  - 当data-type为sorted-set，且schema-syntax为map时，除主键字段外，只能有一个非主键字段，且需要为map类型，同时该字段的map的value需要为double类型，表示score，该字段的map的key表示redis的set中的值。
  - 当data-type为sorted-set，且schema-syntax为array-scores时，除主键字段外，只能有两个非主键字段，且这两个字段的类型需要为array.

    两个字段其中第一个字段类型是array表示Redis的set中的值，第二个字段类型为array<double>，表示相应索引的score。其示例如下：

```
CREATE TABLE redisSink (
  order_id string,
  arrayField Array<String>,
  arrayScore array<double>,
  primary key (order_id) not enforced
```

 版权所有 © 华为技术有限公司

```
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
  "default-score" = '3',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'array-scores'
);
```

语法格式

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'redis',
  'host' = ''
);
```

参数说明

表 2-65 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'redis'。
host	是	无	String	redis连接地址。
port	否	6379	Integer	redis连接端口。
password	否	无	String	redis认证密码。
namespace	否	无	String	redis key的namespace。 例如设置该值为"person"，假设key为"jack"则redis中会是"person:jack"。
delimiter	否	:	String	redis的key和namespace之间的分隔符。
data-type	否	hash	String	redis的数据类型，有下列选项，与redis的数据类型相对应： • hash • list • set • sorted-set • string data-type取值约束详见**data-type取值约束**说明。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
schema-syntax	否	fields	String	redis的schema语义，包含以下值： • fields：适用于所有数据类型。fields类型是指可以设置多个字段，写入时会取每个字段的值。 • fields-scores：适用于sorted set数据类型，表示对每个字段都设置一个字段作为其独立的score。 • array：适用于list、set、sorted set数据类型 • array-scores：适用于sorted set数据类型 • map：适用于hash、sorted set数据类型。 schema-syntax取值约束详见**schema-syntax取值约束**说明。
deploy-mode	否	standalone	String	redis集群的部署模式，支持standalone、master-replica、cluster，默认standalone。 该值可参考redis集群的实例类型介绍。
retry-count	否	5	Integer	连接redis集群的尝试次数。
connection-timeout-millis	否	10000	Integer	尝试连接redis集群时的最大超时时间。
commands-timeout-millis	否	2000	Integer	等待操作完成响应的最大时间。
rebalancing-timeout-millis	否	15000	Integer	redis集群失败时的休眠时间。
default-score	否	0	Double	当data-type设置为“sorted-set”数据类型的默认score。
ignore-retraction	否	false	Boolean	是否忽略retract消息。
skip-null-values	否	true	Boolean	是否跳过null。如果为false，则设置为字符串"null"。
ignore-retractions	否	false	Boolean	连接器应忽略更新插入/撤回流模式下的收回消息。
key-column	否	无	String	Redis 表schema的key

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
sink.delivery-guarantee	否	at-least-once	String	• exactly-once： 记录只传送一次，在故障转移方案下也是如此。如果要生成完整的exactly-once管道，需要源和接收器支持exactly-once，并且已正确配置。 • at-least-once： 确保传递记录，但可能会多次传递同一记录。通常，这种比exactly-once模式更快。 • none： 记录将尽最大努力交付。这通常是处理记录的最快方法，但可能会发生记录丢失或重复的情况。
sink.parallelism	否	无	int	定义接收器的自定义并行度。默认情况下，如果未定义此选项，则规划器将通过考虑全局配置来单独派生每个语句的并行度。
key-ttl-mode	否	no-ttl	String	key-ttl-mode是开启Redis sink TTL的功能参数，key-ttl-mode的限制为：no-ttl、expire-msec、expire-at-date、expire-at-timestamp。 • no-ttl：不设置过期时间。 • expire-msec：设置key多久过期，参数为long类型字符串，单位为毫秒。 • expire-at-date：设置key到某个时间点过期，参数为UTC时间。 • expire-at-timestamp：设置key到某个时间点过期，参数为时间戳。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
key-ttl	否	无	String	key-ttl是key-ttl-mode的补充参数，有以下几种参数值： • 当key-ttl-mode取值为no-ttl时，不需要配置此参数。 • 当key-ttl-mode取值为expire-msec时，需要配置为可以解析成Long型的字符串。例如5000，表示5000ms后key过期。 • 当key-ttl-mode取值为expire-at-date时，需要配置为Date类型字符串，例如2011-12-03T10:15:30，表示到期时间为北京时间2011-12-03 18:15:30。 • 当key-ttl-mode取值为expire-at-timestamp时，需要配置为timestamp类型字符串，单位为毫秒。例如1679385600000，表示到期时间为2023-03-21 16:00:00。

示例

该示例是从Kafka数据源中读取数据，并写入Redis到结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，根据Redis所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Redis的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据redis的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   如下脚本中的加粗参数请根据实际环境修改。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
```


版权所有 © 华为技术有限公司

```
);
--如下redisSink表data-type为默认值hash，schema-syntax定义为fields，将order_id定义为主键，即将该字段的值作为redis的key
CREATE TABLE redisSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '<yourRedis>',
  'password' = '<yourPassword>',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields'
);

insert into redisSink select * from orders;
```

4. 连接Kafka集群，向Kafka中插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

5. 在Redis中分别执行以下命令，查看运行结果：
   - 获取key为"202103241606060001"的结果。

     执行命令：

     ```
     HGETALL 202103241606060001
     ```

     运行结果：

     ```
     1) "user_id"
     2) "0001"
     3) "user_name"
     4) "Alice"
     5) "pay_amount"
     6) "200.0"
     7) "real_pay"
     8) "180.0"
     9) "order_time"
     10) "2021-03-24 16:06:06"
     11) "area_id"
     12) "330106"
     13) "order_channel"
     14) "appShop"
     15) "pay_time"
     16) "2021-03-24 16:10:06"
     ```

   - 获取key为"202103241000000001"的结果。

     执行命令：

     ```
     HGETALL 202103241000000001
     ```

     运行结果：

     ```
     1) "user_id"
     2) "0001"
     3) "user_name"
     4) "Alice"
     5) "pay_amount"
     6) "100.0"
     7) "real_pay"
     ```

 版权所有 © 华为技术有限公司

```
 8) "100.0"
 9) "order_time"
10) "2021-03-24 10:00:00"
11) "area_id"
12) "330106"
13) "order_channel"
14) "webShop"
15) "pay_time"
16) "2021-03-24 10:02:03"
```

常见问题

- Q：当data-type为set时，最终结果数据相比输入数据个数少了是什么原因?

  A：这是因为输入数据中有重复数据，导致在Redis的set中会进行排重，因此个数变少了。

- Q：如果Flink作业的日志中有如下报错信息，应该怎么解决?

  ```
  org.apache.flink.table.api.ValidationException: SQL validation failed. From line 1, column 40 to line 1, column 105: Parameters must be of the same type
  ```

  A：则考虑使用了array类型，但是array中各个字段的类型不统一，需要保持Redis中array中各个字段的类型统一。

- Q：如果Flink作业的日志中有如下报错信息，应该怎么解决?

  ```
  org.apache.flink.addons.redis.core.exception.RedisConnectorException: Wrong Redis schema for 'map' syntax: There should be a key (possibly) and 1 MAP non-key column.
  ```

  A：schema-syantax为map时，在flink中的建表语句只能有一个非主键的列，且该列类型需要为map。

- Q：如果Flink作业的日志中有如下报错信息，应该怎么解决?

  ```
  org.apache.flink.addons.redis.core.exception.RedisConnectorException: Wrong Redis schema for 'array' syntax: There should be a key (possibly) and 1 ARRAY non-key column.
  ```

  A：schema-syantax为array时，在flink中的建表语句只能有一个非主键的列，且该列类型需要为array。

- Q：data-type已经设置了类型，那么schema-syntax的作用是什么?

  A：schema-syntax实际是对特殊类型的处理，如对map和array类型的处理。

  - 对于fields，会对每个字段的值进行处理；对于array和map则会将该字段中的每个元素进行处理。当是fields时，会将该map或array类型的字段值直接作为一个redis中的一个value。
  - 而当是array或者map时，会将array中的每个值作为redis中的一个value，会将map中该字段的value作为redis中的value。array-scores用于sorted-set的data-type，表示使用两个array字段，第一个字段为set中的值，第二个字段表示相应值所对应的score。fields-scores用于sorted-set的data-type，表示从定义的字段中获取score，该类型表示除主键外的奇数字段表示set中的值，该字段的下一个字段表示该字段的score，因此该字段的下一个字段需要为double类型。

- Q：当data-type为hash时，schema-syntax为fields和map的区别是什么?

  A：当使用fields时，会将flink中的字段名作为redis的hash数据类型的field，该字段对应的值作为redis的hash数据类型的value。而当使用map时，会将flink中该字段值的key作为redis的hash数据类型的field，该字段值的value作为redis hash数据类型的value。其具体示例如下：

  - 对于fields：

    i. 创建的Flink作业运行脚本如下：

    ```
    CREATE TABLE orders (
      order_id string,
      order_channel string,
      order_time string,
    ```

```
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE redisSink (
  order_id string,
  maptest Map<string, String>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields'
);

insert into redisSink select order_id, Map[user_id, area_id] from orders;
```

ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop",
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
"area_id":"330106"}
```

iii. 在Redis中，查看其结果如下：

```
1) "maptest"
2) "{0001=330106}"
```

- 对于map：

i. 对于map而言，创建的Flink作业运行脚本如下：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE redisSink (
  order_id string,
  maptest Map<string, String>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'deploy-mode' = '***master-replica***',
  'schema-syntax' = 'map'
```

 版权所有 © 华为技术有限公司

```
);

insert into redisSink select order_id, Map[user_id, area_id] from orders;
```

ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop",
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
"area_id":"330106"}
```

iii. 在Redis中，查看其结果如下：

```
1) "0001"
2) "330106"
```

- Q：当data-type为list时，schema-syntax为fields和array的区别是什么？

  A：fields和array的不同不会导致结果不同。只是在flink建表语句中不同，fields可以是多个字段，而array需要该字段为array类型，且array中的数据类型必须相同，因此fields会更加灵活。

  - 对于fields：

    i. 对于fields而言，创建的Flink作业运行脚本如下：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE redisSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'list',
  'deploy-mode' = '***master-replica***',
  'schema-syntax' = 'fields'
);

insert into redisSink select * from orders;
```

ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop",
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
"area_id":"330106"}
```

 版权所有 © 华为技术有限公司

iii. 使用以下命令查看其结果如下：

Redis执行以下命令：

```
LRANGE 202103241000000001 0 8
```

查询命令执行结果：

```
1) "webShop"
2) "2021-03-24 10:00:00"
3) "100.0"
4) "100.0"
5) "2021-03-24 10:02:03"
6) "0001"
7) "Alice"
8) "330106"
```

- 对于array：

i. 对于array而言，创建的Flink作业运行脚本如下：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE redisSink (
  order_id string,
  arraytest Array<String>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'list',
  'deploy-mode' = '***master-replica***',
  'schema-syntax' = 'array'
);

insert into redisSink select order_id,
array[order_channel,order_time,pay_time,user_id,user_name,area_id] from orders;
```

ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop",
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
"area_id":"330106"}
```

iii. 在Redis中，查看其结果如下（与fields结果不同是因为这里array类型，在flink中的sink建表语句中没有加入double类型的数据，因此少了两个值，并不是由于fields与array不同导致）：

```
1) "webShop"
2) "2021-03-24 10:00:00"
3) "2021-03-24 10:02:03"
4) "0001"
5) "Alice"
6) "330106"
```

 版权所有 © 华为技术有限公司

2.5.16.3 Redis 维表

功能描述

创建Redis表作为维表用于与输入流连接，从而生成相应的宽表。

前提条件

- 要建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- 如果需要获取key的值，则可以通过在flink中设置主键获取，主键字段即对应redis的key。
- 如果定义主键，则不能够定义复合主键，即主键只能是一个字段，不能是多个字段。
- schema-syntax取值约束：
  - 当schema-syntax为map或array时，非主键字段最多只能只有一个，且需要为相应的map或array类型。
  - 当schema-syntax为fields-scores时，非主键字段个数需要为偶数，且除主键字段外，每两个字段的第二个字段的类型需要为double，会将该字段的值视为前一个字段的score，其示例如下：

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  score1 double,
  order_channel string,
  score2 double,
  order_time string,
  score3 double,
  pay_amount double,
  score4 double,
  real_pay double,
  score5 double,
  pay_time string,
  score6 double,
  user_id string,
  score7 double,
  user_name string,
  score8 double,
  area_id string,
  score9 double,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
```


版权所有 © 华为技术有限公司

```
 'deploy-mode' = 'master-replica',
 'schema-syntax' = 'fields-scores'
);
```

- data-type取值约束：
  - 当data-type为set时，flink中定义的非主键字段的类型必须相同。
  - 当data-type为sorted-set且schema-syntax为fields和array时，只能读取redis的sorted set中的值，而不能读取score。
  - 当data-type为string时，只能有一个非主键字段。
  - 当data-type为sorted-set，且schema-syntax为map时，除主键字段外，只能有一个非主键字段，且需要为map类型，同时该字段的map的value需要为double类型，表示score，该字段的map的key表示redis的set中的值。
  - 当data-type为sorted-set，且schema-syntax为array-scores时，除主键字段外，只能有两个非主键字段，且这两个字段的类型需要为array。

    两个字段其中第一个字段类型是array表示Redis的set中的值，第二个字段类型为array<double>，表示相应索引的score。其示例如下：

```
CREATE TABLE redisSink (
 order_id string,
 arrayField Array<String>,
 arrayScore array<double>,
 primary key (order_id) not enforced
) WITH (
 'connector' = 'redis',
 'host' = 'RedisIP',
 'password' = 'RedisPassword',
 'data-type' = 'sorted-set',
 "default-score" = '3',
 'deploy-mode' = 'master-replica',
 'schema-syntax' = 'array-scores'
);
```

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
  ,PRIMARY KEY (attr_name, ...) NOT ENFORCED
)
with (
  'connector' = 'redis',
  'host' = ''
);
```

参数说明

表 2-66 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'redis'。
host	是	无	String	redis连接地址。
port	否	6379	Integer	redis连接端口。
password	否	无	String	redis认证密码。

 版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
namespace	否	无	String	redis key的namespace
delimiter	否	:	String	redis的key和namespace之间的分隔符。
data-type	否	hash	String	redis的数据类型，有下列选项 • hash • list • set • sorted-set • string data-type取值约束详见**data-type取值约束**说明。
schema-syntax	否	fields	String	redis的schema语义，包含以下值： • fields：适用于所有数据类型 • fields-scores：适用于sorted set数据类型 • array：适用于list、set、sorted set数据类型 • array-scores：适用于sorted set数据类型 • map：适用于hash、sorted set数据类型 schema-syntax取值约束详见**schema-syntax取值约束**说明。
deploy-mode	否	standalone	String	redis集群的部署模式，支持standalone、master-replica、cluster，默认standalone。
retry-count	是	5	Integer	设置每个连接请求的队列大小。如果超过队列大小，则命令调用将导致RedisException。将requestQueueSize设置为较低的值将导致在过载期间或连接处于断开状态时更早出现异常。更高的值意味着达到边界需要更长的时间，但可能会有更多的请求排队，并使用更多的堆空间。默认请设置为2147483647。
connection-timeout-millis	否	10000	Integer	尝试连接redis集群时的最大超时时间。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
commands-timeout-millis	否	2000	Integer	等待操作完成响应的最大时间。
rebalancing-timeout-millis	否	15000	Integer	redis集群失败时的休眠时间。
scan-keys-count	否	1000	Integer	每次扫描时读取的数量。
default-score	否	0	Double	当data-type设置为“sorted-set”数据类型的默认score。
deserialize-error-policy	否	fail-job	Enum	数据解析失败时的处理方式。 枚举类型，包含以下值： ● fail-job：作业失败 ● skip-row：跳过当前数据 ● null-field：设置当前数据为null
skip-null-values	否	true	Boolean	是否跳过null。
lookup.async	否	false	Boolean	作为redis维表时，是否使用异步I/O。
lookup.parallelism	否	无	int	定义查找连接运算符的自定义并行度。默认情况下，如果未定义此选项，则规划器将通过考虑全局配置（如果定义了选项“lookup.parallelism”）来推导并行度，否则将考虑输入运算符的并行度。
lookup.batch.interval	否	1s	Duration	批量查找连接可以使用最大延迟来缓冲输入记录。批量查找连接可以使用最大延迟来缓冲输入记录。
lookup.batch.size	否	100L	long	可以缓冲的最大输入记录数，以便进行批量查找连接。
lookup.batch	否	false	Boolean	指定是否启用批量查找优化。如果启用，用户必须同时设置 lookup.batch.interval 和 lookup.batch.size 选项。此外，由于底层批处理间隔干扰机制的实现，用户必须在 flink 配置中显式启用 table.exec.batch-lookup.enabled' 选项
ignore-retractions	否	false	Boolean	连接器应忽略更新插入/撤回流模式下的收回消息。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
key-column	否	无	String	Redis 表schema的key

示例

从Kafka源表中读取数据，将Redis表作为维表，并将二者生成的宽表信息写入Kafka结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，根据Redis和Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Redis和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Redis的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 登录Redis客户端，通过如下命令向Redis发送如下数据：

```
HMSET 330102  area_province_name a1 area_province_name b1 area_county_name c1
area_street_name d1 region_name e1

HMSET 330106  area_province_name a1 area_province_name b1 area_county_name c2
area_street_name d2 region_name e1

HMSET 330108  area_province_name a1 area_province_name b1 area_county_name c3
area_street_name d3 region_name e1

HMSET 330110  area_province_name a1 area_province_name b1 area_county_name c4
area_street_name d4 region_name e1
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。该作业脚本将Kafka为数据源，Redis作为维表，数据写入到Kafka结果表中。

   如下脚本中的加粗参数请根据实际环境修改。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaSourceTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

--创建地址维表
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string,
    primary key (area_id) not enforced -- redis的key
```


版权所有 © 华为技术有限公司

```
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'hash',
  'deploy-mode' = '***master-replica***'
);

--根据地址维表生成详细的包含地址的订单信息宽表
create table order_detail(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) with (
  'connector' = 'kafka',
  'topic' = '***kafkaSinkTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'format' = 'json'
);

insert into order_detail
    select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
           area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
           area.area_street_name, area.region_name  from orders
    left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

5. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"appShop", "order_time":"2021-03-25 15:05:05",
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",
"user_name":"Cindy", "area_id":"330108"}
```

6. 连接Kafka集群，在Kafka的sink topic读取数据，结果数据参考如下：

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_ci
ty_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}

{"order_id":"202103251505050001","order_channel":"appshop","order_time":"2021-03-25
15:05:05","pay_amount":500.0,"real_pay":400.0,"pay_time":"2021-03-25
15:10:00","user_id":"0003","user_name":"Cindy","area_id":"330108","area_province_name":"a1","area_c
ity_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

 版权所有 © 华为技术有限公司

常见问题

如果在windows环境中向redis中写入中文时，会导致写入数据异常，请避免此情况。

2.5.17 Upsert Kafka

功能描述

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。Upsert Kafka 连接器支持以upsert方式从Kafka topic中读取数据并将数据写入Kafka topic。表类型支持源表和结果表。

- 作为source，upsert-kafka 连接器生产changelog流，其中每条数据记录代表一个更新或删除事件。

  数据记录中的value被解释为同一key的最后一个value的UPDATE，如果有这个key（如果不存在相应的key，则该更新被视为INSERT）。用表来类比，changelog流中的数据记录被解释为UPSERT，也称为INSERT/UPDATE，因为任何具有相同key的现有行都被覆盖。另外，value为空的消息将会被视作为DELETE消息。
- 作为sink，upsert-kafka连接器可以消费changelog流。它会将INSERT/UPDATE_AFTER数据作为正常的Kafka消息写入，并将DELETE数据以value为空的Kafka消息写入（表示对应 key 的消息被删除）。Flink将根据主键列的值对数据进行分区，从而保证主键上的消息有序，因此同一主键上的更新/删除消息将落在同一分区中。

表 2-67 支持类别

类别	详情
支持表类型	源表、结果表

前提条件

该场景作业需要运行在DLI的独享队列上，要与kafka集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 认证用的username和password等硬编码到代码中或者明文存储都有很大的安全风险，建议使用DEW管理凭证。配置文件或者环境变量中密文存放，使用时解密，确保安全。**Flink Opensource SQL使用DEW管理访问凭据**
- Upsert Kafka 始终以upsert方式工作，并且需要在DDL中定义主键。在具有相同主键值的消息按序存储在同一个分区的前提下，在 changlog source 定义主键意

版权所有 © 华为技术有限公司

味着 在物化后的 changelog 上主键具有唯一性。定义的主键将决定哪些字段出现在Kafka消息的key中。

- 由于该连接器以 upsert 的模式工作，该连接器作为 source 读入时，可以确保具有相同主键值下仅最后一条消息会生效。
- 数据类型的使用，请参考**Format**章节。

语法格式

```
create table kafkaTable(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'upsert-kafka',
  'topic' = '',
  'properties.bootstrap.servers' = '',
  'key.format' = '',
  'value.format' = ''
);
```

参数说明

表 2-68 参数说明

参数	是否必选	默认参数	数据类型	说明
connector	是	无	String	connector类型，对于upsert kafka连接器，需配置为'upsert-kafka'。
topic	是	无	String	Kafka topic名。
properties.bootstrap.servers	是	无	String	Kafka brokers地址，以逗号分隔。
key.format	是	无	String	用于对Kafka消息中key部分序列化和反序列化的格式。key字段由PRIMARY KEY语法指定。支持的格式如下： • csv • json • avro 请参考**Format**页面以获取更多详细信息和格式参数。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
key.fields-prefix	否	无	String	为键格式的所有字段定义自定义前缀，以避免与值格式的字段发生名称冲突。 默认情况下，前缀为空。如果定义了自定义前缀，则表架构和'key.fields'都将使用前缀名称。在构造密钥格式的数据类型时，将删除前缀，并在密钥格式中使用无前缀的名称。请注意，此选项要求'value.fields-include' 必须设置为'EXCEPT_KEY'。
value.format	是	无	String	用于对 Kafka消息中 value 部分序列化和反序列化的格式。支持的格式： ● csv ● json ● avro 请参考**Format**页面以获取更多详细信息和格式参数。
value.fields-include	是	ALL	String	控制哪些字段应该出现在值中。取值范围如下： ● ALL：消息的value部分将包含schema的所有字段，包括定义中键的字段。 ● EXCEPT_KEY：记录的value部分包含schema的所有内容，定义为主键的字段除外。
properties.*	否	无	String	该选项可以传递任意的Kafka参数。 “properties.”后的后缀名必须匹配定义在 **kafka参数文档**中的参数名。 Flink会自动移除选项名中的 "properties." 前缀，并将转换后的键名以及值传入KafkaClient。 例如：您可以通过'properties.allow.auto.create.topics' = 'false' 来禁止自动创建 topic。 但是'key.deserializer' 和'value.deserializer' 是不允许通过该方式传递参数，因为Flink会重写这些参数的值。
sink.parallelism	否	无	Integer	定义upsert-kafka sink 算子的并行度。默认情况下，由框架确定并行度，与上游连接算子的并行度保持一致。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
sink.buffer-flush.max-rows	否	0	Integer	缓存刷新前，最多能缓存的记录条数。 当sink收到很多同key上的更新时，缓存将保留同 key 的最后一条记录，因此sink缓存能帮助减少发往Kafka topic的数据量，以及避免发送潜在的tombstone消息。可以通过设置为'0'来禁用它。 默认情况下，该选项是未开启的。如果要开启 sink 缓存，需要同时设置'sink.buffer-flush.max-rows'和'sink.buffer-flush.interval'两个选项为大于零的值。
sink.buffer-flush.interval	否	0	Duration	缓存刷新的间隔时间，超过该时间后异步线程将刷新缓存数据。单位可以为毫秒（ms）、秒（s）、分钟（min）或小时（h）。例如'sink.buffer-flush.interval'='10 ms'。 默认情况下，该选项是未开启的。如果要开启 sink 缓存，需要同时设置'sink.buffer-flush.max-rows'和'sink.buffer-flush.interval'两个选项为大于零的值。

元数据

可用的元数据字段列表，请参阅**Kafka连接器**。

示例

- **示例1：该示例是从DMS Kafka数据源中读取数据，并写入到Print结果表中。**
  a. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
  b. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。
  c. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

  注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改。**

```
CREATE TABLE upsertKafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
```


版权所有 © 华为技术有限公司

```
  area_id string,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' =  'KafkaAddress1:KafkaPort,KafkAddress2:KafkaPort',
  'key.format' = 'csv',
  'value.format' = 'json'
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'print'
);

INSERT INTO printSink SELECT * FROM upsertKafkaSource;
```

d. 向Kafka中的指定topic中插入如下数据（注意：**kafka插入数据时请指定key**）。

```
{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202303251505050001", "order_channel":"appshop", "order_time":"2023-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2023-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}

{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330111"}
```

e. 查看taskmanager的out文件，数据结果参考如下：

```
+I(202303251202020001,miniAppShop,2023-03-2512:02:02,60.0,60.0,2023-03-2512:03:00,0002,Bob,330110)
+I(202303251505050001,appshop,2023-03-25 15:05:05,500.0,400.0,2023-03-2515:10:00,0003,Cindy,330108)
-U(202303251202020001,miniAppShop,2023-03-2512:02:02,60.0,60.0,2023-03-2512:03:00,0002,Bob,330110)
+U(202303251202020001,miniAppShop,2023-03-2512:02:02,60.0,60.0,2023-03-2512:03:00,0002,Bob,330111)
```

- **示例2：从Kafka源表获取DMS Kafka source topic数据，通过Upsert Kafka结果表将Kafka source topic数据写入到Kafka sink topic中。**

  a. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。

  b. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。

  c. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

  注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

版权所有 © 华为技术有限公司

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE upsertKafkaSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY(order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' =  '***KafkaAddress1:KafkaPort,KafkAddress2:KafkaPort***',
  'key.format' = 'csv',
  'value.format' = 'json'
);

insert into upsertKafkaSink select * from orders;
```

d. 连接Kafka集群，kafka中source topic发送如下测试数据：

```
{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202303251505050001", "order_channel":"appshop", "order_time":"2023-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2023-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}

{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330111"}
```

e. 连接Kafka集群，获取kafka sink topic的数据，结果参考如下：

```
{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202303251505050001", "order_channel":"appshop", "order_time":"2023-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2023-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}

{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330111"}
```

- **示例3：MRS集群开启Kerberos认证，并且Kafka使用SASL_PLAINTEXT协议，从Kafka源表获取数据，并写入到Print结果表中。**

 版权所有 © 华为技术有限公司

a. 参考**增强型跨源连接**，根据MRS集群所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。

b. 设置MRS集群的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功，否则表示未成功。

c. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改。**

```
CREATE TABLE upsertKafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY(order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' =  '***KafkaAddress1:KafkaPort,KafkAddress2:KafkaPort***',
  'key.format' = 'csv',
  'value.format' = 'json',
  'properties.sasl.mechanism' = 'GSSAPI',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.kerberos.service.name' = 'kafka', --mrs中配置
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = '**username**', --用户名
  'properties.connector.kerberos.krb5' = '**obs://xx/krb5.conf**', --krb5_conf路径
  'properties.connector.kerberos.keytab' = '**obs://xx/user.keytab**' --keytab路径
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'print'
);

INSERT INTO printSink SELECT * FROM upsertKafkaSource;
```

d. 向Kafka中的指定topic中插入如下数据（注意：**kafka插入数据时请指定key**）：

```
{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202303251505050001", "order_channel":"appshop", "order_time":"2023-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2023-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}

{"order_id":"202303251202020001", "order_channel":"miniAppShop", "order_time":"2023-03-25
```


版权所有 © 华为技术有限公司

```
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2023-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330111"}
```

e. 查看taskmanager的out文件，数据结果参考如下：

```
+I(202303251202020001,miniAppShop,2023-03-2512:02:02,60.0,60.0,2023-03-2512:03:00,0002,B
ob,330110)
+I(202303251505050001,appshop,2023-03-2515:05:05,500.0,400.0,2023-03-2515:10:00,0003,Cind
y,330108)
-
U(202303251202020001,miniAppShop,2023-03-2512:02:02,60.0,60.0,2023-03-2512:03:00,0002,B
ob,330110)
+U(202303251202020001,miniAppShop,2023-03-2512:02:02,60.0,60.0,2023-03-2512:03:00,0002,
Bob,330111)
```

常见问题

无

2.6 数据操作语句 DML

2.6.1 SELECT

SELECT

语法格式

```
SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

SELECT语句用于从表中选取数据。

ALL表示返回所有结果。

DISTINCT表示返回不重复结果。

注意事项

- 所查询的表必须是已经存在的表，否则会出错。
- WHERE关键字指定查询的过滤条件，过滤条件中支持算术运算符，关系运算符，逻辑运算符。
- GROUP BY指定分组的字段，可以单字段分组，也可以多字段分组。

示例

找出数量超过3的订单。

```
insert into temp SELECT  * FROM Orders WHERE units > 3;
```

插入一组常量数据。

```
insert into temp select 'Lily’, ‘male’, ‘student’, 17;
```

WHERE 过滤子句

语法格式

```
SELECT  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
```

语法说明

利用WHERE子句过滤查询结果。

注意事项

- 所查询的表必须是已经存在的，否则会出错。
- WHERE条件过滤，将不满足条件的记录过滤掉，返回满足要求的记录。

示例

找出数量超过3并且小于10的订单。

```
insert into temp SELECT  * FROM Orders
  WHERE units > 3 and units < 10;
```

HAVING 过滤子句

功能描述

利用HAVING子句过滤查询结果。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤，HAVING子句支持算术运算，聚合函数等。

注意事项

如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。

示例

根据字段name对表student进行分组，再按组将score最大值大于95的记录筛选出来。

```
insert into temp SELECT name, max(score) FROM student
  GROUP BY name
  HAVING max(score) >95;
```

按列 GROUP BY

功能描述

按列进行分组操作。

语法格式

 版权所有 © 华为技术有限公司

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
```

语法说明

GROUP BY：按列可分为单列GROUP BY与多列GROUP BY。

- 单列GROUP BY：指GROUP BY子句中仅包含一列。
- 多列GROUP BY：指GROUP BY子句中不止一列，查询语句将按照GROUP BY的所有字段分组，所有字段都相同的记录将被放在同一组中。

注意事项

GroupBy在流处理表中会产生更新结果

示例

根据score及name两个字段对表student进行分组，并返回分组结果。

```
insert into temp SELECT name,score, max(score) FROM student
  GROUP BY name,score;
```

表达式 GROUP BY

功能描述

按表达式对流进行分组操作。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
```

语法说明

groupItem：可以是单字段，多字段，也可以是字符串函数等调用，不能是聚合函数。

注意事项

无

示例

先利用substring函数取字段name的子字符串，并按照该子字符串进行分组，返回每个子字符串及对应的记录数。

```
insert into temp SELECT substring(name,6),count(name) FROM student
  GROUP BY substring(name,6);
```

Grouping sets, Rollup, Cube

功能描述

- GROUPING SETS 的 GROUP BY 子句可以生成一个等效于由多个简单 GROUP BY 子句的 UNION ALL 生成的结果集，并且其效率比 GROUP BY 要高。
- ROLLUP与CUBE按一定的规则产生多种分组，然后按各种分组统计数据。
- CUBE生成的结果集显示了所选列中值的所有组合的聚合。

 版权所有 © 华为技术有限公司

- Rollup生成的结果集显示了所选列中值的某一层次结构的聚合。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY groupingItem]
```

语法说明

groupingItem：是Grouping sets(columnName [, columnName]*)、Rollup(columnName [, columnName]*)、Cube(columnName [, columnName]*)

注意事项

无

示例

分别产生基于user和product的结果

```
INSERT INTO temp SELECT SUM(amount)
FROM Orders
GROUP BY GROUPING SETS ((user), (product));
```

GROUP BY 中使用 HAVING 过滤

功能描述

利用HAVING子句在表分组后实现过滤。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。

注意事项

- 如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。HAVING与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。
- HAVING中除聚合函数外所使用的字段必须是GROUP BY中出现的字段。
- HAVING子句支持算术运算，聚合函数等。

示例

先依据num对表transactions进行分组，再利用HAVING子句对查询结果进行过滤，price与amount乘积的最大值大于5000的记录将被筛选出来，返回对应的num及price与amount乘积的最大值。

```
insert into temp SELECT num, max(price*amount) FROM transactions
  WHERE time > '2016-06-01'
  GROUP BY num
  HAVING max(price*amount)>5000;
```

 版权所有 © 华为技术有限公司

2.6.2 INSERT INTO

本节操作介绍使用INSERT INTO 语句将作业结果写入Sink表中。

写数据至一个 Sink 表

- **语法格式**

```
INSERT INTO your_sink
SELECT ... FROM your_source WHERE ...
```

- **示例**

本例定义了两个表my_source 和my_sink，并使用INSERT INTO语句source表选择数据并插入到sink表。

```
--使用datagen connector创建源表my_source
CREATE TABLE my_source (
  name VARCHAR,
  age BIGINT
) WITH (
  'connector' = 'datagen');

--使用jdbc connector创建目标表my_sink
CREATE TABLE my_sink (
  name VARCHAR,
  age BIGINT
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://xxx/your-database',
  'table-name' = 'your-table',
  'username' = 'your-username',
  'password' = 'your-password'
);

--使用INSERT INTO语句从my_source表选择数据，并插入到my_sink表
INSERT INTO my_sink
SELECT name, age
FROM my_source;
```

写数据至多个 Sink 表

EXECUTE STATEMENT SET BEGIN ... END; 是写数据至多个Sink表的必填语句，用于定义在同一个作业中执行多个插入数据的操作。

⚠ 注意

写数据至多个Sink表时，**EXECUTE STATEMENT SET BEGIN ... END;**是必填项。

- **语法格式**

```
EXECUTE STATEMENT SET BEGIN
  -- 第一个DML语句
  INSERT INTO your_sink1
  SELECT ... FROM your_source WHERE ...;

  -- 第二个DML语句
  INSERT INTO your_sink2
  SELECT ... FROM your_source WHERE ...

...
END;
```

- **示例**

版权所有 © 华为技术有限公司

本例定义了源表datagen_source、Sink表print_sinkA和print_sinkB。然后使用EXECUTE STATEMENT执行两个INSERT INTO语句，分将转换后的数据写入两个不同的sink。

```
--使用 datagen connector创建源表 datagen_source
CREATE TABLE datagen_source (
  name VARCHAR,
  age BIGINT
) WITH (
  'connector' = 'datagen'
);

--使用print connector创建结果表 print_sinkA 和 print_sinkB

CREATE TABLE print_sinkA(
  name VARCHAR,
  age BIGINT
) WITH (
  'connector' = 'print'
);

CREATE TABLE print_sinkB(
  name VARCHAR,
  age BIGINT
) WITH (
  'connector' = 'print'
);

--使用 EXECUTE STATEMENT SET BEGIN来执行两个 INSERT INTO 语句。
--第一个INSERT INTO语句将datagen_source表中的数据按需转换后写入 print_sinkA。
--第二个 INSERT INTO 语句将数据按需转换后写入 print_sinkB。。
EXECUTE  STATEMENT SET BEGIN
INSERT INTO print_sinkA
  SELECT UPPER(name), min(age)
  FROM datagen_source
  GROUP BY UPPER(name);
INSERT INTO print_sinkB
  SELECT LOWER(name), max(age)
  FROM datagen_source
  GROUP BY LOWER(name);
END;
```

2.6.3 集合操作

Union/Union ALL/Intersect/Except

语法格式

```
query UNION [ ALL ] | Intersect | Except query
```

语法说明

- UNION返回多个查询结果的并集。
- Intersect返回多个查询结果的交集。
- Except返回多个查询结果的差集。

注意事项

- 集合运算是以一定条件将表首尾相接，所以其中每一个SELECT语句返回的列数必须相同，列的类型一定要相同，列名不一定要相同。
- UNION默认是去重的，UNION ALL是不去重的。

示例

输出Orders1和Orders2的并集，不包含重复记录。

 版权所有 © 华为技术有限公司

```
insert into temp SELECT  * FROM Orders1
  UNION SELECT  * FROM Orders2;
```

IN

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  WHERE column_name IN (value (, value)* ) | query
```

语法说明

IN操作符允许在where子句中规定多个值。如果表达式在给定的表子查询中存在，则返回 true 。

注意事项

子查询表必须由单个列构成，且该列的数据类型需与表达式保持一致。

示例

输出Orders中NewProducts中product的user和amount信息。

```
insert into temp SELECT user, amount
FROM Orders
WHERE product IN (
    SELECT product FROM NewProducts
);
```

2.6.4 窗口

GROUP WINDOW

语法说明

Group Window定义在GROUP BY里，每个分组只输出一条记录，包括以下几种：

- 分组函数

> **⚠ 注意**
>
> 在流处理表中的 SQL 查询中，分组窗口函数的 time_attr 参数必须引用一个合法的时间属性，且该属性需要指定行的处理时间或事件时间。
>
> - time_attr设置为event-time时参数类型为timestamp(3)类型。
> - time_attr设置为processing-time时无需指定类型。
>
> 对于批处理的 SQL 查询，分组窗口函数的 time_attr 参数必须是一个timestamp类型的属性。

版权所有 © 华为技术有限公司

表 2-69 分组函数表

分组窗口函数	说明
TUMBLE(time_attr, interval)	定义一个滚动窗口。 滚动窗口把行分配到有固定持续时间（ interval ）的不重叠的连续窗口。 例如，5 分钟的滚动窗口以 5 分钟为间隔对行进行分组。 滚动窗口可以定义在事件时间（批处理、流处理）或处理时间（流处理）上。
HOP(time_attr, interval, interval)	定义一个跳跃的时间窗口（在 Table API 中称为滑动窗口）。 滑动窗口有一个固定的持续时间（ 第二个 interval 参数 ）以及一个滑动的间隔（第一个 interval 参数 ）。 如果滑动间隔小于窗口的持续时间，滑动窗口则会出现重叠；因此，行将会被分配到多个窗口中。 例如，一个大小为 15 分钟的滑动窗口，其滑动间隔为 5 分钟，将会把每一行数据分配到 3 个 15 分钟的窗口中。滑动窗口可以定义在事件时间（批处理、流处理）或处理时间（流处理）上。
SESSION(time_attr, interval)	定义一个会话时间窗口。 会话时间窗口没有一个固定的持续时间，但是它们的边界会根据 interval 所定义的不活跃时间所确定；即一个会话时间窗口在定义的间隔时间内没有事件出现，该窗口会被关闭。 例如时间窗口的间隔时间是 30 分钟，当其不活跃的时间达到30分钟后，如果观测到新的记录，则会启动一个新的会话时间窗口（否则该行数据会被添加到当前的窗口），且如果在 30 分钟内没有观测到新记录，这个窗口将会被关闭。会话时间窗口可以使用事件时间（批处理、流处理）或处理时间（流处理）。

- 窗口辅助函数

  可以使用以下辅助函数选择组窗口的开始和结束时间戳以及时间属性。

  注意

  辅助函数必须使用与GROUP BY 子句中的分组窗口函数完全相同的参数来调用。

版权所有 © 华为技术有限公司

表 2-70 窗口辅助函数表

辅助函数	说明
TUMBLE_START(time_attr, interval) HOP_START(time_attr, interval, interval) SESSION_START(time_attr, interval)	返回相对应的滚动、滑动和会话窗口范围内的下界时间戳。
TUMBLE_END(time_attr, interval) HOP_END(time_attr, interval, interval) SESSION_END(time_attr, interval)	返回相对应的滚动、滑动和会话窗口*范围以外*的上界时间戳。 范围以外的上界时间戳不可以 在随后基于时间的操作中，作为行时间属性使用，比如基于时间窗口的join以及分组窗口或分组窗口上的聚合。
TUMBLE_ROWTIME(time_attr, interval) HOP_ROWTIME(time_attr, interval, interval) SESSION_ROWTIME(time_attr, interval)	返回的是一个可用于后续需要基于时间的操作的时间属性（rowtime attribute），比如基于时间窗口的join以及 分组窗口或分组窗口上的聚合。
TUMBLE_PROCTIME(time_attr, interval) HOP_PROCTIME(time_attr, interval, interval) SESSION_PROCTIME(time_attr, interval)	返回一个可用于后续需要基于时间的操作的 处理时间参数，比如基于时间窗口的join以及分组窗口或分组窗口上的聚合.

示例

```
// 每天计算SUM（金额）（事件时间）。
insert into temp SELECT name,
    TUMBLE_START(ts, INTERVAL '1' DAY) as wStart,
    SUM(amount)
    FROM Orders
    GROUP BY TUMBLE(ts, INTERVAL '1' DAY), name;

// 每天计算SUM（金额）（处理时间）。
insert into temp SELECT name,
    SUM(amount)
    FROM Orders
    GROUP BY TUMBLE(proctime, INTERVAL '1' DAY), name;

// 每个小时计算事件时间中最近24小时的SUM（数量）。
insert into temp SELECT product,
    SUM(amount)
    FROM Orders
    GROUP BY HOP(ts, INTERVAL '1' HOUR, INTERVAL '1' DAY), product;

// 计算每个会话的SUM（数量），间隔12小时的不活动间隙（事件时间）。
insert into temp SELECT name,
    SESSION_START(ts, INTERVAL '12' HOUR) AS sStart,
    SESSION_END(ts, INTERVAL '12' HOUR) AS sEnd,
    SUM(amount)
```


版权所有 © 华为技术有限公司

```
FROM Orders
GROUP BY SESSION(ts, INTERVAL '12' HOUR), name;
```

TUMBLE WINDOW 扩展

功能描述

DLI TUMBLE函数功能增强主要包括以下功能：

- TUMBLE窗口周期性触发，控制延迟

  TUMBLE窗口结束之前，可以根据设置的触发频率周期性地触发窗口，输出从窗口开始时间到当前周期时间窗口内的计算结果值，但不影响最终窗口输出值，从而在窗口结束前的每个周期都可以看到最新的结果。

- 提高数据的精确性

  在窗口结束后，允许设置延迟时间。根据设置的延迟时间，每到达一个迟到数据，则更新窗口的输出结果

注意事项

- 如果使用insert语句将结果写入sink中，则sink需要支持upsert模式，所以结果表需要支持upsert操作，且定义主键。
- 延迟时间设置仅用于事件时间，在处理时间中不生效。
- 辅助函数必须使用与 GROUP BY 子句中的分组窗口函数完全相同的参数来调用。
- 如果使用事件时间，则需要使用watermark标识，代码如下（其中order_time被标识为事件时间列，watermark时间设置为3秒）：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  watermark for order_time as order_time - INTERVAL '3' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
```

- 如果使用处理时间，则需要使用计算列设置，其代码如下（其中proc即为处理时间列）：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proc as proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
```

 版权所有 © 华为技术有限公司

```
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
```

语法格式

```
TUMBLE(time_attr, window_interval, period_interval, lateness_interval)
```

语法示例

例如当前time_attr属性列为：testtime，窗口时间间隔为10秒，设置延迟时间为10秒
语法示例为：

```
TUMBLE(testtime, INTERVAL '10' SECOND, INTERVAL '10' SECOND, INTERVAL '10' SECOND)
```

参数说明

表 2-71 参数说明

参数	说明	参数格式
time_attr	表示相应的事件时间或者处理时间属性列。 ● time_attr设置为event-time时参数类型为timestamp(3)类型。 ● time_attr设置为processing-time时无需指定类型。	-
window_interval	表示窗口的持续时长。	● 格式1：**INTERVAL** '10' **SECOND** 表示窗口时间间隔为10秒，请根据实际情况修改该时间值。 ● 格式2：**INTERVAL** '10' **MINUTE** 表示窗口时间间隔为10分钟，请根据实际情况修改该时间值。 ● 格式3：**INTERVAL** '10' **DAY** 表示窗口时间间隔为10天，请根据实际情况修改该时间值。
period_interval	表示在窗口范围内周期性触发的频率，即在窗口结束前，从窗口开启开始，每隔period_interval时长更新一次输出结果。如果没有设置，则默认没有使用周期触发策略。	
lateness_interval	表示窗口结束后延迟lateness_interval时长，继续统计在窗口结束后延迟时间内到达的属于该窗口的数据，而且在延迟时间内到达的每个数据都会更新输出结果。 **说明** 当时间窗口为处理时间时，无论lateness_interval为何值，都不会有效果。	

说明

period_interval和lateness_interval不可为负数。

- 当period_interval为0时，表示没有使用窗口的周期触发策略；
- 当lateness_interval为0时，表示没有使用窗口结束后的延迟策略；
- 当二者都没有填写时，默认两种策略都没有配置，仅使用普通的TUMBLE窗口。
- 如果仅需使用延迟时间策略，则需要将上述period_interval格式中的'10'设置为 '0'。

辅助函数

表 2-72 辅助函数

辅助函数	说明
TUMBLE_START(time_attr, window_interval, period_interval, lateness_interval)	返回相对应的滚动窗口范围内的下界时间戳。
TUMBLE_END(time_attr, window_interval, period_interval, lateness_interval)	返回相对应的滚动窗口范围以外的上界时间戳。

示例

1. 根据订单信息使用kafka作为数据源表，JDBC作为数据结果表统计用户在30秒内的订单数量，并根据窗口的订单id和窗口开启时间作为主键，将结果实时统计到JDBC中：

步骤1 根据MySQL和kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据MySQL和kafka的地址测试队列连通性。如果能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 在MySQL的flink数据库下创建表order_count，创建语句如下：

```
CREATE TABLE `flink`.`order_count` (
    `user_id` VARCHAR(32) NOT NULL,
    `window_start` TIMESTAMP NOT NULL,
    `window_end` TIMESTAMP NULL,
    `total_num` BIGINT UNSIGNED NULL,
    PRIMARY KEY (`user_id`, `window_start`)
)   ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4
    COLLATE = utf8mb4_general_ci;
```

步骤3 创建flink opensource sql作业，并提交运行作业（这里设置窗口的大小为30秒，触发周期为10秒，延迟时间设置为5秒，即窗口结束前如果结果有更新，则每隔十秒输出一次中间结果。在watermark到达使得窗口结束后，事件时间在watermark5秒内的数据仍然会被处理，并统计到当前所属窗口；如果在5秒以外，则该数据会被丢弃）：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  watermark for order_time as order_time - INTERVAL '3' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE jdbcSink (
  user_id string,
  window_start timestamp(3),
  window_end timestamp(3),
```


版权所有 © 华为技术有限公司

```
  total_num BIGINT,
  primary key (user_id, window_start) not enforced
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://<yourMySQL>:3306/flink',
  'table-name' = 'order_count',
  'username' = '<yourUserName>',
  'password' = '<yourPassword>',
  'sink.buffer-flush.max-rows' = '1'
);

insert into jdbcSink select
    order_id,
    TUMBLE_START(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5' SECOND),
    TUMBLE_END(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5' SECOND),
    COUNT(*) from orders
    GROUP BY user_id, TUMBLE(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5'
SECOND);
```

步骤4 向kafka中插入数据（这里假设同一个用户在不同时间下的订单，且因为某种原因导致10:00:13的订单数据较晚到达）：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000002", "order_channel":"webShop", "order_time":"2021-03-24 10:00:20",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000003", "order_channel":"webShop", "order_time":"2021-03-24 10:00:33",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000004", "order_channel":"webShop", "order_time":"2021-03-24 10:00:13",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

步骤5 在MySQL中使用下述语句查看输出结果,，输出结果如下（因无法展示周期性输出结果，所以这里展示的是最终结果）：

```
select * from order_count
user_id     window_start         window_end         total_num
0001      2021-03-24 10:00:00  2021-03-24 10:00:30    3
0001      2021-03-24 10:00:30  2021-03-24 10:01:00    1
```

----结束

OVER WINDOW

Over Window与Group Window区别在于Over window每一行都会输出一条记录。

语法格式

```
SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  ROWS
 BETWEEN (UNBOUNDED|rowCOUNT) PRECEDING AND CURRENT ROW FROM TABLENAME

SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  RANGE
  BETWEEN (UNBOUNDED|timeInterval) PRECEDING AND CURRENT ROW FROM TABLENAME
```

语法说明

 版权所有 © 华为技术有限公司

表 2-73 参数说明

参数	参数说明
PARTITION BY	指定分组的主键，每个分组各自进行计算。
ORDER BY	指定数据按processing time或event time作为时间戳。
ROWS	个数窗口。
RANGE	时间窗口。

注意事项

- 所有的聚合必须定义到同一个窗口中，即相同的分区、排序和区间。
- 当前仅支持 PRECEDING (无界或有界) 到 CURRENT ROW 范围内的窗口、FOLLOWING 所描述的区间并未支持。
- ORDER BY 必须指定于单个的时间属性。

示例

```
// 计算从规则启动到目前为止的计数及总和(in proctime)
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as
cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt2
    FROM Orders;

// 计算最近四条记录的计数及总和(in proctime)
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt2
    FROM Orders;

// 计算最近60s的计数及总和(in eventtime),基于事件时间处理，事件时间为Orders中的timeattr字段。
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60'
SECOND PRECEDING AND CURRENT ROW) as cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND
PRECEDING AND CURRENT ROW) as cnt2
    FROM Orders;
```

2.6.4.1 窗口函数

窗口表值函数（Windowing TVFs）

窗口是处理无限流的核心。窗口把流分割为有限大小的 “桶”，这样就可以在其之上进行计算。

Apache Flink 提供了如下 **窗口表值函数（table-valued function, 缩写TVF）**把表的数据划分到窗口中：

- 滚动窗口
- 滑动窗口

 版权所有 © 华为技术有限公司

- 累积窗口

逻辑上，每个元素可以应用于一个或多个窗口，这取决于所使用的窗口表值函数的类型。例如：滑动窗口可以把单个元素分配给多个窗口。

窗口表值函数 是 Flink 定义的多态表函数（Polymorphic Table Function，缩写 PTF），PTF 是 SQL 2016 标准中的一种特殊的表函数，它可以把表作为一个参数。

窗口表值函数是分组函数（已废弃）的替代方案。窗口表值函数 更符合 SQL 标准，在支持基于窗口的复杂计算上也更强大。例如：窗口 TopN、窗口 Join。而分组窗口函数只支持窗口聚合。

更多介绍和使用请参考开源社区文档：**窗口函数**。

窗口函数简介

Apache Flink 提供3个内置的窗口表值函数：TUMBLE，HOP 和 CUMULATE。

窗口表值函数的返回值包括原生列和附加的三个指定窗口的列，分别是：“window_start”，“window_end”，“window_time”。

在批计算模式，window_time 是 TIMESTAMP 或者 TIMESTAMP_LTZ 类型（具体哪种类型取决于输入的时间字段类型）的字段。 window_time 字段用于后续基于时间的操作，例如：其他的窗口表值函数，或者interval joins，over aggregations。 它的值总是等于 window_end - 1ms。

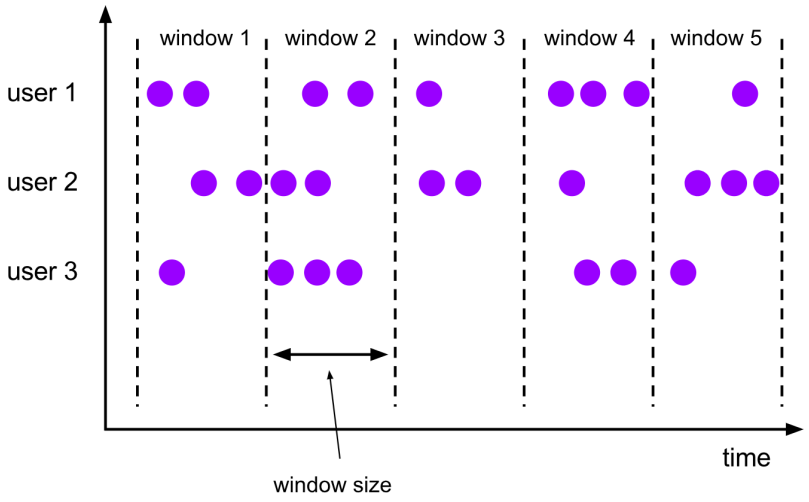
滚动窗口（TUMBLE）

- **功能描述**

  TUMBLE函数指定每个元素到一个指定大小的窗口中。滚动窗口的大小固定且不重复。

  例如：假设指定了一个 5 分钟的滚动窗口。Flink 将每 5 分钟生成一个新的窗口。

  图 2-3 滚动窗口示例图



- **语法描述**

版权所有 © 华为技术有限公司

TUMBLE 函数通过时间属性字段为每行数据分配一个窗口。 在流计算模式，时间属性字段必须被指定为事件或处理时间属性。 在批计算模式，窗口表函数的时间属性字段必须是 TIMESTAMP 或 TIMESTAMP_LTZ 的类型。

TUMBLE 的返回值包括原始表的所有列和附加的三个用于指定窗口的列，分别是：“window_start”，“window_end”，“window_time”。函数运行后，原有的时间属性 “timecol” 将转换为一个常规的 timestamp 列。

```
TUMBLE(TABLE data, DESCRIPTOR(timecol), size [, offset ])
```

表 2-74 TUMBLE 函数参数说明

参数	是否必选	说明
data	是	拥有时间属性列的表。
timecol	是	列描述符，决定数据的哪个时间属性列应该映射到窗口。
size	是	窗口的大小（时长）
offset	否	窗口的偏移量。

- **示例**

```
-- tables must have time attribute, e.g. `bidtime` in this table
Flink SQL> desc Bid;
+-------------+------------------------+------+-----+--------+--------------------------------+
|        name |                   type | null | key | extras |                      watermark |
+-------------+------------------------+------+-----+--------+--------------------------------+
|     bidtime | TIMESTAMP(3) *ROWTIME* | true |     |        | `bidtime` - INTERVAL '1' SECOND |
|       price |          DECIMAL(10, 2) | true |     |        |                                |
|        item |                 STRING | true |     |        |                                |
+-------------+------------------------+------+-----+--------+--------------------------------+

Flink SQL> SELECT * FROM Bid;
+------------------+-------+------+
|          bidtime | price | item |
+------------------+-------+------+
| 2020-04-15 08:05 |  4.00 | C    |
| 2020-04-15 08:07 |  2.00 | A    |
| 2020-04-15 08:09 |  5.00 | D    |
| 2020-04-15 08:11 |  3.00 | B    |
| 2020-04-15 08:13 |  1.00 | E    |
| 2020-04-15 08:17 |  6.00 | F    |
+------------------+-------+------+

Flink SQL> SELECT * FROM TABLE(
  TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES));
-- or with the named params
-- note: the DATA param must be the first
Flink SQL> SELECT * FROM TABLE(
  TUMBLE(
    DATA => TABLE Bid,
    TIMECOL => DESCRIPTOR(bidtime),
    SIZE => INTERVAL '10' MINUTES));
+------------------+-------+------+------------------+------------------+-------------------------+
|          bidtime | price | item |     window_start |       window_end |             window_time |
+------------------+-------+------+------------------+------------------+-------------------------+
| 2020-04-15 08:05 |  4.00 | C    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:07 |  2.00 | A    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:09 |  5.00 | D    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
| 2020-04-15 08:17 |  6.00 | F    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
+------------------+-------+------+------------------+------------------+-------------------------+
```

 版权所有 © 华为技术有限公司

```
-- apply aggregation on the tumbling windowed table
Flink SQL> SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
+------------------+------------------+-------+
|     window_start |       window_end | price |
+------------------+------------------+-------+
| 2020-04-15 08:00 | 2020-04-15 08:10 | 11.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 | 10.00 |
+------------------+------------------+-------+
```

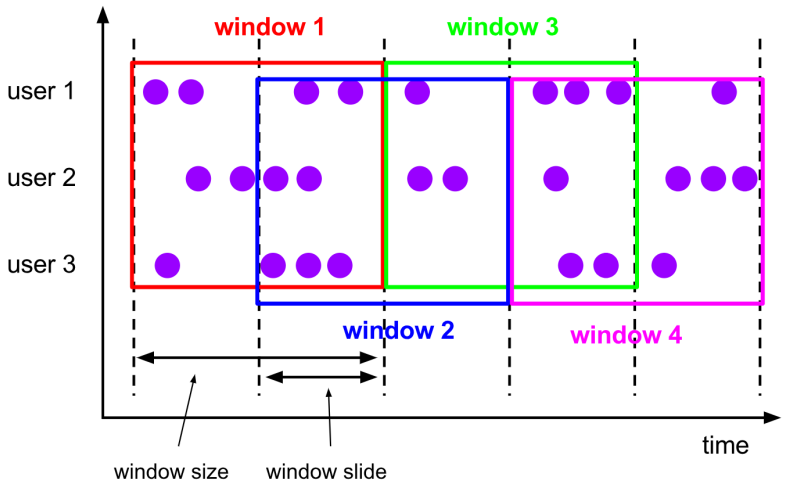
滑动窗口（HOP）

- **功能描述**

  滑动窗口函数指定元素到一个定长的窗口中。和滚动窗口很像，有窗口大小参数，另外增加了一个窗口滑动步长参数。如果滑动步长小于窗口大小，就能产生数据重叠的效果。在这个例子里，数据可以被分配在多个窗口。

  例如：可以定义一个每5分钟滑动一次。大小为10分钟的窗口。每5分钟获得最近10分钟到达的数据的窗口,如下图所示：

  图 2-4 滑动窗口示例图

- **语法描述**

  HOP 函数通过时间属性字段为每一行数据分配了一个窗口。 在流计算模式，这个时间属性字段必须被指定为事件或处理时间属性。在批计算模式，这个窗口表函数的时间属性字段必须是 TIMESTAMP 或 TIMESTAMP_LTZ 的类型。

  HOP 的返回值包括原始表的所有列和附加的三个用于指定窗口的列，分别是：“window_start”，“window_end”，“window_time”。函数运行后，原有的时间属性 “timecol” 将转换为一个常规的 timestamp 列。

  ```
  HOP(TABLE data, DESCRIPTOR(timecol), slide, size [, offset ])
  ```

  表 2-75 HOP 函数参数说明

参数	是否必选	说明
data	是	拥有时间属性列的表。

版权所有 © 华为技术有限公司

参数	是否必选	说明
timecol	是	列描述符，决定数据的哪个时间属性列应该映射到窗口。
slide	是	窗口的滑动步长。
size	是	窗口的大小（时长）
offset	否	窗口的偏移量。

- **示例**

```
> SELECT * FROM TABLE(
    HOP(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '5' MINUTES, INTERVAL '10' MINUTES));
-- or with the named params
-- note: the DATA param must be the first
> SELECT * FROM TABLE(
    HOP(
      DATA => TABLE Bid,
      TIMECOL => DESCRIPTOR(bidtime),
      SLIDE => INTERVAL '5' MINUTES,
      SIZE => INTERVAL '10' MINUTES));
+------------------+-------+------+------------------+------------------+-------------------------+
|          bidtime | price | item |     window_start |       window_end |             window_time |
+------------------+-------+------+------------------+------------------+-------------------------+
| 2020-04-15 08:05 |  4.00 | C    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:05 |  4.00 | C    | 2020-04-15 08:05 | 2020-04-15 08:15 | 2020-04-15 08:14:59.999 |
| 2020-04-15 08:07 |  2.00 | A    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:07 |  2.00 | A    | 2020-04-15 08:05 | 2020-04-15 08:15 | 2020-04-15 08:14:59.999 |
| 2020-04-15 08:09 |  5.00 | D    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:09 |  5.00 | D    | 2020-04-15 08:05 | 2020-04-15 08:15 | 2020-04-15 08:14:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:05 | 2020-04-15 08:15 | 2020-04-15 08:14:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:05 | 2020-04-15 08:15 | 2020-04-15 08:14:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
| 2020-04-15 08:17 |  6.00 | F    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
| 2020-04-15 08:17 |  6.00 | F    | 2020-04-15 08:15 | 2020-04-15 08:25 | 2020-04-15 08:24:59.999 |
+------------------+-------+------+------------------+------------------+-------------------------+


-- apply aggregation on the hopping windowed table
> SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    HOP(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '5' MINUTES, INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
+------------------+------------------+-------+
|     window_start |       window_end | price |
+------------------+------------------+-------+
| 2020-04-15 08:00 | 2020-04-15 08:10 | 11.00 |
| 2020-04-15 08:05 | 2020-04-15 08:15 | 15.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 | 10.00 |
| 2020-04-15 08:15 | 2020-04-15 08:25 |  6.00 |
+------------------+------------------+-------+
```

累积窗口（CUMULATE）

- 功能描述

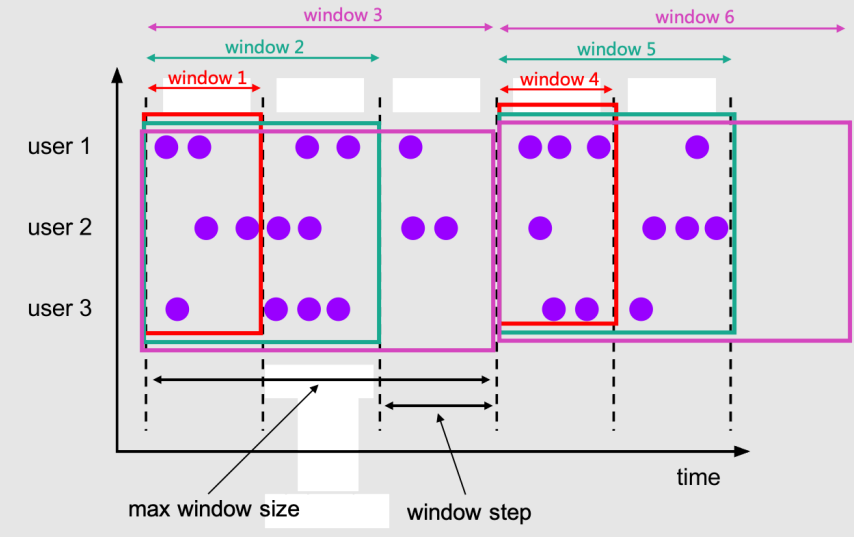
  累积窗口在某些场景中非常有用，比如说提前触发的滚动窗口。例如：每日仪表盘从 00:00 开始每分钟绘制累积 UV，10:00 时 UV 就是从 00:00 到 10:00 的UV总数。累积窗口可以简单且有效地实现它。

  CUMULATE 函数指定元素到多个窗口，从初始的窗口开始，直到达到最大的窗口大小的窗口，所有的窗口都包含其区间内的元素，另外，窗口的开始时间是固定的。 您可以将 CUMULATE 函数视为首先应用具有最大窗口大小的 TUMBLE 窗

口，然后将每个滚动窗口拆分为具有相同窗口开始但窗口结束步长不同的几个窗口。 所以累积窗口会产生重叠并且没有固定大小。

例如：1小时步长，24小时大小的累计窗口，每天可以获得如下这些窗口：[00:00, 01:00)，[00:00, 02:00)，[00:00, 03:00)， …， [00:00, 24:00)

图 2-5 累积窗口示例图



- **语法描述**

  CUMULATE 函数通过时间属性字段为每一行数据分配了一个窗口。 在流计算模式，这个时间属性字段必须被指定为事件或处理时间属性。在批计算模式，这个窗口表函数的时间属性字段必须是 TIMESTAMP 或 TIMESTAMP_LTZ 的类型。

  CUMULATE 的返回值包括原始表的所有列和附加的三个用于指定窗口的列，分别是：“window_start”，“window_end”，“window_time”。函数运行后，原有的时间属性 “timecol” 将转换为一个常规的 timestamp 列。

```
CUMULATE(TABLE data, DESCRIPTOR(timecol), step, size)
```

表 2-76 CUMULATE 函数参数说明

参数	是否必选	说明
data	是	拥有时间属性列的表。
timecol	是	列描述符，决定数据的哪个时间属性列应该映射到窗口。
step	是	指定连续的累积窗口之间增加的窗口大小。
size	是	窗口的大小（时长）
offset	否	窗口的偏移量。

- **示例**

```
> SELECT * FROM TABLE(
    CUMULATE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '2' MINUTES, INTERVAL '10' MINUTES));
-- or with the named params
-- note: the DATA param must be the first
```


版权所有 © 华为技术有限公司

```
> SELECT * FROM TABLE(
   CUMULATE(
     DATA => TABLE Bid,
     TIMECOL => DESCRIPTOR(bidtime),
     STEP => INTERVAL '2' MINUTES,
     SIZE => INTERVAL '10' MINUTES));
+------------------+-------+------+------------------+------------------+-------------------------+
|          bidtime | price | item |     window_start |       window_end |             window_time  |
+------------------+-------+------+------------------+------------------+-------------------------+
| 2020-04-15 08:05 |  4.00 | C    | 2020-04-15 08:00 | 2020-04-15 08:06 | 2020-04-15 08:05:59.999 |
| 2020-04-15 08:05 |  4.00 | C    | 2020-04-15 08:00 | 2020-04-15 08:08 | 2020-04-15 08:07:59.999 |
| 2020-04-15 08:05 |  4.00 | C    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:07 |  2.00 | A    | 2020-04-15 08:00 | 2020-04-15 08:08 | 2020-04-15 08:07:59.999 |
| 2020-04-15 08:07 |  2.00 | A    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:09 |  5.00 | D    | 2020-04-15 08:00 | 2020-04-15 08:10 | 2020-04-15 08:09:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:10 | 2020-04-15 08:12 | 2020-04-15 08:11:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:10 | 2020-04-15 08:14 | 2020-04-15 08:13:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:10 | 2020-04-15 08:16 | 2020-04-15 08:15:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:10 | 2020-04-15 08:18 | 2020-04-15 08:17:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:10 | 2020-04-15 08:14 | 2020-04-15 08:13:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:10 | 2020-04-15 08:16 | 2020-04-15 08:15:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:10 | 2020-04-15 08:18 | 2020-04-15 08:17:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
| 2020-04-15 08:17 |  6.00 | F    | 2020-04-15 08:10 | 2020-04-15 08:18 | 2020-04-15 08:17:59.999 |
| 2020-04-15 08:17 |  6.00 | F    | 2020-04-15 08:10 | 2020-04-15 08:20 | 2020-04-15 08:19:59.999 |
+------------------+-------+------+------------------+------------------+-------------------------+

-- apply aggregation on the cumulating windowed table
> SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    CUMULATE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
+------------------+------------------+-------+
|     window_start |       window_end | price |
+------------------+------------------+-------+
| 2020-04-15 08:00 | 2020-04-15 08:06 |  4.00 |
| 2020-04-15 08:00 | 2020-04-15 08:08 |  6.00 |
| 2020-04-15 08:00 | 2020-04-15 08:10 | 11.00 |
| 2020-04-15 08:10 | 2020-04-15 08:12 |  3.00 |
| 2020-04-15 08:10 | 2020-04-15 08:14 |  4.00 |
| 2020-04-15 08:10 | 2020-04-15 08:16 |  4.00 |
| 2020-04-15 08:10 | 2020-04-15 08:18 | 10.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 | 10.00 |
+------------------+------------------+-------+
```

窗口偏移

Offset 可选参数，可以用来改变窗口的分配。可以是正或者负的区间。默认情况下窗口的偏移是 0。不同的偏移值可以决定记录分配的窗口。 例如：在 10 分钟大小的滚动窗口下，时间戳为 2021-06-30 00:00:04 的数据会被分配到哪个窗口呢?

- 当 offset 为 -16 MINUTE，数据会分配到窗口 [2021-06-29 23:54:00, 2021-06-30 00:04:00)。
- 当 offset 为 -6 MINUTE，数据会分配到窗口 [2021-06-29 23:54:00, 2021-06-30 00:04:00)。
- 当 offset 为 -4 MINUTE，数据会分配到窗口 [2021-06-29 23:56:00, 2021-06-30 00:06:00)。
- 当 offset 为 0，数据会分配到窗口 [2021-06-30 00:00:00, 2021-06-30 00:10:00)。
- 当 offset 为 4 MINUTE，数据会分配到窗口 [2021-06-29 23:54:00, 2021-06-30 00:04:00)。

 版权所有 © 华为技术有限公司

- 当 offset 为 6 MINUTE，数据会分配到窗口 [2021-06-29 23:56:00, 2021-06-30 00:06:00)。
- 当 offset 为 16 MINUTE，数据会分配到窗口 [2021-06-29 23:56:00, 2021-06-30 00:06:00)。 我们可以发现，有些不同的窗口偏移参数对窗口分配的影响是一样的。在上面的例子中，-16 MINUTE，-6 MINUTE 和 4 MINUTE 对 10 分钟大小的滚动窗口效果相同。

说明

窗口偏移只影响窗口的分配，并不会影响 Watermark

```
-- NOTE: Currently Flink doesn't support evaluating individual window table-valued function,
--  window table-valued function should be used with aggregate operation,
--  this example is just used for explaining the syntax and the data produced by table-valued function.
Flink SQL> SELECT * FROM TABLE(
   TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES, INTERVAL '1' MINUTES));
-- or with the named params
-- note: the DATA param must be the first
Flink SQL> SELECT * FROM TABLE(
   TUMBLE(
     DATA => TABLE Bid,
     TIMECOL => DESCRIPTOR(bidtime),
     SIZE => INTERVAL '10' MINUTES,
     OFFSET => INTERVAL '1' MINUTES));
+------------------+-------+------+------------------+------------------+-------------------------+
|          bidtime | price | item |     window_start |       window_end |             window_time  |
+------------------+-------+------+------------------+------------------+-------------------------+
| 2020-04-15 08:05 |  4.00 | C    | 2020-04-15 08:01 | 2020-04-15 08:11 | 2020-04-15 08:10:59.999 |
| 2020-04-15 08:07 |  2.00 | A    | 2020-04-15 08:01 | 2020-04-15 08:11 | 2020-04-15 08:10:59.999 |
| 2020-04-15 08:09 |  5.00 | D    | 2020-04-15 08:01 | 2020-04-15 08:11 | 2020-04-15 08:10:59.999 |
| 2020-04-15 08:11 |  3.00 | B    | 2020-04-15 08:11 | 2020-04-15 08:21 | 2020-04-15 08:20:59.999 |
| 2020-04-15 08:13 |  1.00 | E    | 2020-04-15 08:11 | 2020-04-15 08:21 | 2020-04-15 08:20:59.999 |
| 2020-04-15 08:17 |  6.00 | F    | 2020-04-15 08:11 | 2020-04-15 08:21 | 2020-04-15 08:20:59.999 |
+------------------+-------+------+------------------+------------------+-------------------------+

-- apply aggregation on the tumbling windowed table
Flink SQL> SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES, INTERVAL '1' MINUTES))
  GROUP BY window_start, window_end;
+------------------+------------------+-------+
|     window_start |       window_end | price |
+------------------+------------------+-------+
| 2020-04-15 08:01 | 2020-04-15 08:11 | 11.00 |
| 2020-04-15 08:11 | 2020-04-15 08:21 | 10.00 |
+------------------+------------------+-------+
```

2.6.4.2 窗口聚合

窗口表值函数(TVF)聚合

窗口聚合是通过GROUP BY子句定义的，其特征是包含**窗口表值函数**产生的“window_start” 和 “window_end” 列。和普通的 GROUP BY 子句一样，窗口聚合对于每个组会计算出一行数据。和其他连续表上的聚合不同，窗口聚合不产生中间结果，只在窗口结束产生一个总的聚合结果，另外，窗口聚合会清除不需要的中间状态。

更多介绍和使用请参考开源社区文档：**窗口聚合**。

说明

分组窗口的开始和结束时间戳可以通过 window_start 和 window_end 来选定。

版权所有 © 华为技术有限公司

- **窗口表值函数**

  Flink 支持在 TUMBLE， HOP 和 CUMULATE 上进行窗口聚合。

  - 在流模式下，窗口表值函数的时间属性字段必须是事件时间或处理时间。关于窗口函数更多信息，参见 **窗口表值函数（Windowing TVFs）**。
  - 在批模式下，窗口表值函数的时间属性字段必须是 TIMESTAMP 或 TIMESTAMP_LTZ 类型的。

```
-- tables must have time attribute, e.g. `bidtime` in this table
Flink SQL> desc Bid;
+-------------+------------------------+------+-----+--------+--------------------------------+
|        name |                   type | null | key | extras |                      watermark |
+-------------+------------------------+------+-----+--------+--------------------------------+
|     bidtime | TIMESTAMP(3) *ROWTIME* | true |     |        | `bidtime` - INTERVAL '1' SECOND |
|       price |          DECIMAL(10, 2) | true |     |        |                                |
|        item |                 STRING | true |     |        |                                |
| supplier_id |                 STRING | true |     |        |                                |
+-------------+------------------------+------+-----+--------+--------------------------------+

Flink SQL> SELECT * FROM Bid;
+------------------+-------+------+-------------+
|          bidtime | price | item | supplier_id |
+------------------+-------+------+-------------+
| 2020-04-15 08:05 | 4.00  | C    | supplier1   |
| 2020-04-15 08:07 | 2.00  | A    | supplier1   |
| 2020-04-15 08:09 | 5.00  | D    | supplier2   |
| 2020-04-15 08:11 | 3.00  | B    | supplier2   |
| 2020-04-15 08:13 | 1.00  | E    | supplier1   |
| 2020-04-15 08:17 | 6.00  | F    | supplier2   |
+------------------+-------+------+-------------+

-- tumbling window aggregation
Flink SQL> SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
+------------------+------------------+-------+
|     window_start |       window_end | price |
+------------------+------------------+-------+
| 2020-04-15 08:00 | 2020-04-15 08:10 | 11.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 | 10.00 |
+------------------+------------------+-------+

-- hopping window aggregation
Flink SQL> SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    HOP(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '5' MINUTES, INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
+------------------+------------------+-------+
|     window_start |       window_end | price |
+------------------+------------------+-------+
| 2020-04-15 08:00 | 2020-04-15 08:10 | 11.00 |
| 2020-04-15 08:05 | 2020-04-15 08:15 | 15.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 | 10.00 |
| 2020-04-15 08:15 | 2020-04-15 08:25 | 6.00  |
+------------------+------------------+-------+

-- cumulative window aggregation
Flink SQL> SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    CUMULATE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
+------------------+------------------+-------+
|     window_start |       window_end | price |
+------------------+------------------+-------+
| 2020-04-15 08:00 | 2020-04-15 08:06 | 4.00  |
| 2020-04-15 08:00 | 2020-04-15 08:08 | 6.00  |
| 2020-04-15 08:00 | 2020-04-15 08:10 | 11.00 |
```


版权所有 © 华为技术有限公司

```
| 2020-04-15 08:10 | 2020-04-15 08:12 | 3.00  |
| 2020-04-15 08:10 | 2020-04-15 08:14 | 4.00  |
| 2020-04-15 08:10 | 2020-04-15 08:16 | 4.00  |
| 2020-04-15 08:10 | 2020-04-15 08:18 | 10.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 | 10.00 |
```

- **GROUPING SETS**

  窗口聚合也支持 GROUPING SETS 语法。Grouping Sets 可以通过一个标准的 GROUP BY 语句来描述更复杂的分组操作。数据按每个指定的 Grouping Sets 分别分组，并像简单的 GROUP BY 子句一样为每个组进行聚合。

  GROUPING SETS 窗口聚合中 GROUP BY 子句必须包含 window_start 和 window_end 列，但 GROUPING SETS 子句中不能包含这两个字段。

```
Flink SQL> SELECT window_start, window_end, supplier_id, SUM(price) as price
  FROM TABLE(
    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end, GROUPING SETS ((supplier_id), ());
+------------------+------------------+-------------+-------+
|     window_start |       window_end | supplier_id | price |
+------------------+------------------+-------------+-------+
| 2020-04-15 08:00 | 2020-04-15 08:10 |      (NULL) | 11.00 |
| 2020-04-15 08:00 | 2020-04-15 08:10 |   supplier2 |  5.00 |
| 2020-04-15 08:00 | 2020-04-15 08:10 |   supplier1 |  6.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 |      (NULL) | 10.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 |   supplier2 |  9.00 |
| 2020-04-15 08:10 | 2020-04-15 08:20 |   supplier1 |  1.00 |
+------------------+------------------+-------------+-------+
```

  GROUPING SETS 的每个子列表可以是空的，多列或表达式，它们的解释方式和直接使用 GROUP BY 子句是一样的。一个空的 Grouping Sets 表示所有行都聚合在一个分组下，即使没有数据，也会输出结果。

  对于 Grouping Sets 中的空子列表，结果数据中的分组或表达式列会用NULL代替。例如，上例中的 GROUPING SETS ((supplier_id), ()) 里的 () 就是空子列表，与其对应的结果数据中的 supplier_id 列使用 NULL 填充。

- **ROLLUP**

  ROLLUP 是一种特定通用类型 Grouping Sets 的简写。代表着指定表达式和所有前缀的列表，包括空列表。

  例如：ROLLUP (one,two) 等效于 GROUPING SET((one,two),(one),()).

  ROLLUP 窗口聚合中 GROUP BY 子句必须包含 window_start 和 window_end 列，但 ROLLUP 子句中不能包含这两个字段。

  例如：下面这个查询和上个例子中的效果是一样的。

```
SELECT window_start, window_end, supplier_id, SUM(price) as price
FROM TABLE(
    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
GROUP BY window_start, window_end, ROLLUP (supplier_id);
```

- **CUBE**

  CUBE 是一种特定通用类型 Grouping Sets 的简写。代表着指定列表以及所有可能的子集和幂集。

  CUBE 窗口聚合中 GROUP BY 子句必须包含 window_start 和 window_end 列，但 CUBE 子句中不能包含这两个字段。

  例如：下面两个查询是等效的。

```
SELECT window_start, window_end, item, supplier_id, SUM(price) as price
  FROM TABLE(
    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end, CUBE (supplier_id, item);

SELECT window_start, window_end, item, supplier_id, SUM(price) as price
  FROM TABLE(
```

 版权所有 © 华为技术有限公司

```
  TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
 GROUP BY window_start, window_end, GROUPING SETS (
    (supplier_id, item),
    (supplier_id      ),
    (             item),
    (                  )
)
```

- **多级窗口聚合**

  window_start 和 window_end 列是普通的时间戳字段，并不是时间属性。因此它们不能在后续的操作中当做时间属性进行基于时间的操作。

  为了传递时间属性，需要在 GROUP BY 子句中添加 window_time 列。window_time 是**窗口表值函数（Windowing TVFs）**产生的三列之一，它是窗口的时间属性。 window_time 添加到 GROUP BY 子句后就能被选定了。下面的查询可以把它用于后续基于时间的操作，比如：多级窗口聚合和Window TopN。

  下面展示了一个多级窗口聚合：第一个窗口聚合后把时间属性传递给第二个窗口聚合。

```
-- tumbling 5 minutes for each supplier_id
CREATE VIEW window1 AS
-- Note: The window start and window end fields of inner Window TVF are optional in the select clause. However, if they appear in the clause, they need to be aliased to prevent name conflicting with the window start and window end of the outer Window TVF.
SELECT window_start as window_5mintumble_start, window_end as window_5mintumble_end, window_time as rowtime, SUM(price) as partial_price
  FROM TABLE(
    TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '5' MINUTES))
  GROUP BY supplier_id, window_start, window_end, window_time;

-- tumbling 10 minutes on the first window
SELECT window_start, window_end, SUM(partial_price) as total_price
  FROM TABLE(
     TUMBLE(TABLE window1, DESCRIPTOR(rowtime), INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
```

2.6.4.3 窗口 Top-N

功能描述

窗口 Top-N 是特殊的 Top-N，它返回每个分区键的每个窗口的N个最小或最大值。

与普通Top-N不同，窗口Top-N只在窗口最后返回汇总的Top-N数据，不会产生中间结果。窗口 Top-N 会在窗口结束后清除不需要的中间状态。

窗口 Top-N 适用于用户不需要每条数据都更新Top-N结果的场景，相对普通Top-N来说性能更好。通常，窗口 Top-N 直接用于**窗口表值函数（Windowing TVFs）**窗口 Top-N 可以用于基于**窗口表值函数（Windowing TVFs）**的操作之上，比如窗口聚合，窗口Top-N和 窗口关联。

窗口 Top-N 的语法和普通的 Top-N 相同。 除此之外，窗口 Top-N 需要 PARTITION BY 子句包含窗口表值函数或窗口聚合产生的 window_start 和 window_end。 否则优化器无法翻译。

更多介绍和使用请参考开源社区文档：**窗口Top-N**。

语法格式

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     ROW_NUMBER() OVER (PARTITION BY window_start, window_end [, col_key1...]
```

```
    ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
  FROM table_name) -- relation applied windowing TVF
WHERE rownum <= N [AND conditions]
```

注意事项

Flink只支持在滚动，滑动和累计窗口表值函数后进行窗口 Top-N。

示例

在窗口聚合后进行窗口 Top-N

下面的示例展示了在10分钟的滚动窗口上计算销售额位列前三的供应商。

```
-- tables must have time attribute, e.g. `bidtime` in this table
Flink SQL> desc Bid;
+-------------+------------------------+------+-----+--------+--------------------------------+
|        name |                   type | null | key | extras |                      watermark |
+-------------+------------------------+------+-----+--------+--------------------------------+
|     bidtime | TIMESTAMP(3) *ROWTIME* | true |     |        | `bidtime` - INTERVAL '1' SECOND |
|       price |          DECIMAL(10, 2) | true |     |        |                                |
|        item |                 STRING | true |     |        |                                |
| supplier_id |                 STRING | true |     |        |                                |
+-------------+------------------------+------+-----+--------+--------------------------------+

Flink SQL> SELECT * FROM Bid;
+------------------+-------+------+-------------+
|          bidtime | price | item | supplier_id |
+------------------+-------+------+-------------+
| 2020-04-15 08:05 |  4.00 |    A |   supplier1 |
| 2020-04-15 08:06 |  4.00 |    C |   supplier2 |
| 2020-04-15 08:07 |  2.00 |    G |   supplier1 |
| 2020-04-15 08:08 |  2.00 |    B |   supplier3 |
| 2020-04-15 08:09 |  5.00 |    D |   supplier4 |
| 2020-04-15 08:11 |  2.00 |    B |   supplier3 |
| 2020-04-15 08:13 |  1.00 |    E |   supplier1 |
| 2020-04-15 08:15 |  3.00 |    H |   supplier2 |
| 2020-04-15 08:17 |  6.00 |    F |   supplier5 |
+------------------+-------+------+-------------+

Flink SQL> SELECT *
  FROM (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY window_start, window_end ORDER BY price DESC) as
rownum
    FROM (
      SELECT window_start, window_end, supplier_id, SUM(price) as price, COUNT(*) as cnt
      FROM TABLE(
        TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
      GROUP BY window_start, window_end, supplier_id
    )
  ) WHERE rownum <= 3;
+------------------+------------------+-------------+-------+-----+--------+
|     window_start |       window_end | supplier_id | price | cnt | rownum |
+------------------+------------------+-------------+-------+-----+--------+
| 2020-04-15 08:00 | 2020-04-15 08:10 |   supplier1 |  6.00 |   2 |      1 |
| 2020-04-15 08:00 | 2020-04-15 08:10 |   supplier4 |  5.00 |   1 |      2 |
| 2020-04-15 08:00 | 2020-04-15 08:10 |   supplier2 |  4.00 |   1 |      3 |
| 2020-04-15 08:10 | 2020-04-15 08:20 |   supplier5 |  6.00 |   1 |      1 |
| 2020-04-15 08:10 | 2020-04-15 08:20 |   supplier2 |  3.00 |   1 |      2 |
| 2020-04-15 08:10 | 2020-04-15 08:20 |   supplier3 |  2.00 |   1 |      3 |
+------------------+------------------+-------------+-------+-----+--------+
```

在窗口表值函数后进行窗口 Top-N

下面的示例展示了在10分钟的滚动窗口上计算价格位列前三的数据。

```
Flink SQL> SELECT *
  FROM (
```

```
   SELECT *, ROW_NUMBER() OVER (PARTITION BY window_start, window_end ORDER BY price DESC) as
rownum
   FROM TABLE(
            TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
  ) WHERE rownum <= 3;
+------------------+-------+------+-------------+------------------+------------------+--------+
|          bidtime | price | item | supplier_id |     window_start |       window_end | rownum |
+------------------+-------+------+-------------+------------------+------------------+--------+
| 2020-04-15 08:05 |  4.00 |    A |   supplier1 | 2020-04-15 08:00 | 2020-04-15 08:10 |      2 |
| 2020-04-15 08:06 |  4.00 |    C |   supplier2 | 2020-04-15 08:00 | 2020-04-15 08:10 |      3 |
| 2020-04-15 08:09 |  5.00 |    D |   supplier4 | 2020-04-15 08:00 | 2020-04-15 08:10 |      1 |
| 2020-04-15 08:11 |  2.00 |    B |   supplier3 | 2020-04-15 08:10 | 2020-04-15 08:20 |      3 |
| 2020-04-15 08:15 |  3.00 |    H |   supplier2 | 2020-04-15 08:10 | 2020-04-15 08:20 |      2 |
| 2020-04-15 08:17 |  6.00 |    F |   supplier5 | 2020-04-15 08:10 | 2020-04-15 08:20 |      1 |
+------------------+-------+------+-------------+------------------+------------------+--------+
```

2.6.4.4 窗口去重

功能描述

窗口去重是一种特殊的去重，它根据指定的多个列来删除重复的行，保留每个窗口和分区键的第一个或最后一个数据。

对于流式查询，与普通去重不同，窗口去重只在窗口的最后返回结果数据，不会产生中间结果。它会清除不需要的中间状态。 因此，窗口去重查询在用户不需要更新结果时，性能较好。通常，窗口去重直接用于窗口表值函数上。另外，它可以用于基于窗口表值函数的操作。比如窗口聚合，窗口TopN和窗口关联。

窗口Top-N的语法和普通的Top-N相同。 除此之外，窗口去重需要 PARTITION BY 子句包含表的 window_start 和 window_end 列。 否则优化器无法翻译。

Flink 使用 ROW_NUMBER() 移除重复数据，就像窗口TopN一样。理论上，窗口是一种特殊的窗口 Top-N：N是1并且是根据处理时间或事件时间排序的。

更多介绍和使用请参考开源社区文档：**窗口去重**。

语法格式

```
SELECT [column_list]
FROM (
  SELECT [column_list],
    ROW_NUMBER() OVER (PARTITION BY window_start, window_end [, col_key1...]
      ORDER BY time_attr [asc|desc]) AS rownum
  FROM table_name) -- relation applied windowing TVF
WHERE (rownum = 1 | rownum <=1 | rownum < 2) [AND conditions]
```

参数说明：

- ROW_NUMBER()：为每一行分配一个唯一且连续的序号，从1开始。
- PARTITION BY window_start, window_end [, col_key1...]： 指定分区字段，需要包含window_start， window_end以及其他分区键。
- ORDER BY time_attr [asc|desc]： 指定排序列，必须是时间属性。目前 Flink 支持处理时间属性和事件时间属性。 Order by ASC 表示保留第一行，Order by DESC 表示保留最后一行。
- WHERE (rownum = 1 | rownum <=1 | rownum < 2)： 优化器通过 rownum = 1 | rownum <=1 | rownum < 2 来识别查询能否被翻译成窗口去重。

 版权所有 © 华为技术有限公司

注意事项

- Flink 只支持在滚动窗口、滑动窗口和累积窗口的窗口表值函数后进行窗口去重
- 窗口去重只支持根据事件时间属性进行排序

示例

本示例展示了在10分钟的滚动窗口上保持最后一条记录。

```
-- tables must have time attribute, e.g. `bidtime` in this table
Flink SQL> DESC Bid;
+-------------+------------------------+------+-----+--------+--------------------------------+
|        name |                   type | null | key | extras |                      watermark |
+-------------+------------------------+------+-----+--------+--------------------------------+
|     bidtime | TIMESTAMP(3) *ROWTIME* | true |     |        | `bidtime` - INTERVAL '1' SECOND |
|       price |          DECIMAL(10, 2) | true |     |        |                                |
|        item |                 STRING | true |     |        |                                |
+-------------+------------------------+------+-----+--------+--------------------------------+

Flink SQL> SELECT * FROM Bid;
+------------------+-------+------+
|          bidtime | price | item |
+------------------+-------+------+
| 2020-04-15 08:05 |  4.00 | C    |
| 2020-04-15 08:07 |  2.00 | A    |
| 2020-04-15 08:09 |  5.00 | D    |
| 2020-04-15 08:11 |  3.00 | B    |
| 2020-04-15 08:13 |  1.00 | E    |
| 2020-04-15 08:17 |  6.00 | F    |
+------------------+-------+------+

Flink SQL> SELECT *
  FROM (
    SELECT bidtime, price, item, supplier_id, window_start, window_end,
      ROW_NUMBER() OVER (PARTITION BY window_start, window_end ORDER BY bidtime DESC) AS
rownum
    FROM TABLE(
            TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
  ) WHERE rownum <= 1;
+------------------+-------+------+-------------+------------------+------------------+--------+
|          bidtime | price | item | supplier_id |     window_start |       window_end | rownum |
+------------------+-------+------+-------------+------------------+------------------+--------+
| 2020-04-15 08:09 |  5.00 |    D |   supplier4 | 2020-04-15 08:00 | 2020-04-15 08:10 |      1 |
| 2020-04-15 08:17 |  6.00 |    F |   supplier5 | 2020-04-15 08:10 | 2020-04-15 08:20 |      1 |
+------------------+-------+------+-------------+------------------+------------------+--------+
```

2.6.4.5 窗口关联

窗口关联就是增加时间维度到关联条件中。在此过程中，窗口关联将两个流中在同一窗口且符合 join 条件的元素 join 起来。窗口关联的语义和 DataStream window join 相同。

在流式查询中，与其他连续表上的关联不同，窗口关联不产生中间结果，只在窗口结束产生一个最终的结果。另外，窗口关联会清除不需要的中间状态。通常，窗口关联和 窗口表值函数 一起使用。而且，窗口关联可以在其他基于 窗口表值函数 的操作后使用，例如 窗口聚合，窗口 Top-N 和 窗口关联。目前，窗口关联需要在 join on 条件中包含两个输入表的 window_start 等值条件和 window_end 等值条件。窗口关联支持 INNER/LEFT/RIGHT/FULL OUTER/ANTI/SEMI JOIN。

更多介绍和使用请参考开源社区文档：**窗口关联**。

注意事项

- 窗口关联需要在 join on 条件中包含两个输入表的 window_start 等值条件和 window_end 等值条件。
- 关联的左右两边必须使用相同的窗口表值函数。
- 窗口关联支持作用在滚动（TUMBLE）、滑动（HOP）和累积（CUMULATE）窗口表值函数之上，但是还不支持会话窗口。

INNER/LEFT/RIGHT/FULL OUTER

INNER/LEFT/RIGHT/FULL OUTER 这几种窗口关联的语法非常相似，我们在这里只举一个 FULL OUTER JOIN 的例子。 当执行窗口关联时，所有具有相同 key 和相同滚动窗口的数据会被关联在一起。这里给出一个基于 TUMBLE Window TVF 的窗口连接的例子。 在下面的例子中，通过将 join 的时间区域限定为固定的 5 分钟，数据集被分成两个不同的时间窗口：[12:00,12:05) 和 [12:05,12:10)。L2 和 R2 不能 join 在一起是因为它们不在一个窗口中。

语法格式

```
SELECT ...
FROM L [LEFT|RIGHT|FULL OUTER] JOIN R -- L and R are relations applied windowing TVF
ON L.window_start = R.window_start AND L.window_end = R.window_end AND ...
```

示例

当执行窗口关联时，所有具有相同 key 和相同滚动窗口的数据会被关联在一起。这里给出一个基于 TUMBLE Window TVF 的窗口连接的例子。 在下面的例子中，通过将 join 的时间区域限定为固定的 5 分钟，数据集被分成两个不同的时间窗口：[12:00,12:05) 和 [12:05,12:10)。L2 和 R2 不能 join 在一起是因为它们不在一个窗口中。

```
Flink SQL> desc LeftTable;
+----------+------------------------+------+-----+--------+---------------------------------+
|     name |                   type | null | key | extras |                       watermark |
+----------+------------------------+------+-----+--------+---------------------------------+
| row_time | TIMESTAMP(3) *ROWTIME* | true |     |        | `row_time` - INTERVAL '1' SECOND |
|      num |                    INT | true |     |        |                                 |
|       id |                 STRING | true |     |        |                                 |
+----------+------------------------+------+-----+--------+---------------------------------+

Flink SQL> SELECT * FROM LeftTable;
+------------------+-----+----+
|         row_time | num | id |
+------------------+-----+----+
| 2020-04-15 12:02 |   1 | L1 |
| 2020-04-15 12:06 |   2 | L2 |
| 2020-04-15 12:03 |   3 | L3 |
+------------------+-----+----+

Flink SQL> desc RightTable;
+----------+------------------------+------+-----+--------+---------------------------------+
|     name |                   type | null | key | extras |                       watermark |
+----------+------------------------+------+-----+--------+---------------------------------+
| row_time | TIMESTAMP(3) *ROWTIME* | true |     |        | `row_time` - INTERVAL '1' SECOND |
|      num |                    INT | true |     |        |                                 |
|       id |                 STRING | true |     |        |                                 |
+----------+------------------------+------+-----+--------+---------------------------------+

Flink SQL> SELECT * FROM RightTable;
+------------------+-----+----+
|         row_time | num | id |
+------------------+-----+----+
| 2020-04-15 12:01 |   2 | R2 |
```

 版权所有 © 华为技术有限公司

```
| 2020-04-15 12:04 |   3 | R3 |
| 2020-04-15 12:05 |   4 | R4 |
+------------------+-----+----+

Flink SQL> SELECT L.num as L_Num, L.id as L_Id, R.num as R_Num, R.id as R_Id,
          COALESCE(L.window_start, R.window_start) as window_start,
          COALESCE(L.window_end, R.window_end) as window_end
          FROM (
             SELECT * FROM TABLE(TUMBLE(TABLE LeftTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
          ) L
          FULL JOIN (
             SELECT * FROM TABLE(TUMBLE(TABLE RightTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
          ) R
          ON L.num = R.num AND L.window_start = R.window_start AND L.window_end = R.window_end;
+-------+------+-------+------+------------------+------------------+
| L_Num | L_Id | R_Num | R_Id |     window_start |       window_end |
+-------+------+-------+------+------------------+------------------+
|     1 |   L1 |  null | null | 2020-04-15 12:00 | 2020-04-15 12:05 |
|  null | null |     2 |   R2 | 2020-04-15 12:00 | 2020-04-15 12:05 |
|     3 |   L3 |     3 |   R3 | 2020-04-15 12:00 | 2020-04-15 12:05 |
|     2 |   L2 |  null | null | 2020-04-15 12:05 | 2020-04-15 12:10 |
|  null | null |     4 |   R4 | 2020-04-15 12:05 | 2020-04-15 12:10 |
+-------+------+-------+------+------------------+------------------+
```

SEMI

如果在同一个窗口中，左侧记录在右侧至少有一个匹配的记录时，半窗口连接（Semi Window Join）就会输出左侧的记录。

```
Flink SQL> SELECT *
          FROM (
             SELECT * FROM TABLE(TUMBLE(TABLE LeftTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
          ) L WHERE L.num IN (
            SELECT num FROM (
             SELECT * FROM TABLE(TUMBLE(TABLE RightTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
            ) R WHERE L.window_start = R.window_start AND L.window_end = R.window_end);
+------------------+-----+----+------------------+------------------+-------------------------+
|         row_time | num | id |     window_start |       window_end |             window_time |
+------------------+-----+----+------------------+------------------+-------------------------+
| 2020-04-15 12:03 |   3 | L3 | 2020-04-15 12:00 | 2020-04-15 12:05 | 2020-04-15 12:04:59.999 |
+------------------+-----+----+------------------+------------------+-------------------------+

Flink SQL> SELECT *
          FROM (
             SELECT * FROM TABLE(TUMBLE(TABLE LeftTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
          ) L WHERE EXISTS (
            SELECT * FROM (
             SELECT * FROM TABLE(TUMBLE(TABLE RightTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
            ) R WHERE L.num = R.num AND L.window_start = R.window_start AND L.window_end =
R.window_end);
+------------------+-----+----+------------------+------------------+-------------------------+
|         row_time | num | id |     window_start |       window_end |             window_time |
+------------------+-----+----+------------------+------------------+-------------------------+
| 2020-04-15 12:03 |   3 | L3 | 2020-04-15 12:00 | 2020-04-15 12:05 | 2020-04-15 12:04:59.999 |
+------------------+-----+----+------------------+------------------+-------------------------+
```

ANTI

反窗口连接（Anti Window Join）是内窗口连接（Inner Window Join）的相反操作：它包含了每个公共窗口内所有未关联上的行。

 版权所有 © 华为技术有限公司

```
Flink SQL> SELECT *
          FROM (
            SELECT * FROM TABLE(TUMBLE(TABLE LeftTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
          ) L WHERE L.num NOT IN (
            SELECT num FROM (
              SELECT * FROM TABLE(TUMBLE(TABLE RightTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
            ) R WHERE L.window_start = R.window_start AND L.window_end = R.window_end);
+------------------+-----+----+------------------+------------------+-------------------------+
|         row_time | num | id |     window_start |       window_end |            window_time  |
+------------------+-----+----+------------------+------------------+-------------------------+
| 2020-04-15 12:02 |   1 | L1 | 2020-04-15 12:00 | 2020-04-15 12:05 | 2020-04-15 12:04:59.999 |
| 2020-04-15 12:06 |   2 | L2 | 2020-04-15 12:05 | 2020-04-15 12:10 | 2020-04-15 12:09:59.999 |
+------------------+-----+----+------------------+------------------+-------------------------+

Flink SQL> SELECT *
          FROM (
            SELECT * FROM TABLE(TUMBLE(TABLE LeftTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
          ) L WHERE NOT EXISTS (
            SELECT * FROM (
              SELECT * FROM TABLE(TUMBLE(TABLE RightTable, DESCRIPTOR(row_time), INTERVAL '5'
MINUTES))
            ) R WHERE L.num = R.num AND L.window_start = R.window_start AND L.window_end =
R.window_end);
+------------------+-----+----+------------------+------------------+-------------------------+
|         row_time | num | id |     window_start |       window_end |            window_time  |
+------------------+-----+----+------------------+------------------+-------------------------+
| 2020-04-15 12:02 |   1 | L1 | 2020-04-15 12:00 | 2020-04-15 12:05 | 2020-04-15 12:04:59.999 |
| 2020-04-15 12:06 |   2 | L2 | 2020-04-15 12:05 | 2020-04-15 12:10 | 2020-04-15 12:09:59.999 |
+------------------+-----+----+------------------+------------------+-------------------------+
```

2.6.5 分组聚合

聚合函数把多行输入数据计算为一行结果。例如，有一些聚合函数可以计算一组行的“COUNT”、“SUM”、“AVG”（平均）、“MAX”（最大）和“MIN”（最小）。

对于流式查询，用于计算查询结果的状态可能无限膨胀。状态的大小大多数情况下取决于去重行的数量和分组持续的时间，持续时间较短的 group 窗口不会产生状态过大的问题。可以提供一个合适的状态 time-to-live (TTL) 配置来防止状态过大。注意：这可能会影响查询结果的正确性。

更多介绍和使用请参考开源社区文档：**分组聚合**。

DISTINCT 聚合

DISTINCT 聚合在聚合函数前去掉重复的数据。下面的示例计算 Orders 表中不同 order_ids 的数量，而不是总行数。

```
SELECT COUNT(DISTINCT order_id) FROM Orders
```

GROUPING SETS

Grouping Sets 可以通过一个标准的 GROUP BY 语句来描述更复杂的分组操作。数据按每个指定的 Grouping Sets 分别分组，并像简单的 group by 子句一样为每个组进行聚合。

GROUPING SETS 的每个子列表可以是：空的，多列或表达式，它们的解释方式和直接使用 GROUP BY 子句是一样的。一个空的 Grouping Sets 表示所有行都聚合在一个分组下，即使没有数据，也会输出结果。

对于 Grouping Sets 中的空子列表，结果数据中的分组或表达式列会用NULL代替。

```
SELECT supplier_id, rating, COUNT(*) AS total
FROM (VALUES
    ('supplier1', 'product1', 4),
    ('supplier1', 'product2', 3),
    ('supplier2', 'product3', 3),
    ('supplier2', 'product4', 4))
AS Products(supplier_id, product_id, rating)
GROUP BY GROUPING SETS ((supplier_id, rating), (supplier_id), ())
```

ROLLUP

ROLLUP 是一种特定通用类型 Grouping Sets 的简写。代表着指定表达式和所有前缀的列表，包括空列表。

```
SELECT supplier_id, rating, COUNT(*)
FROM (VALUES
    ('supplier1', 'product1', 4),
    ('supplier1', 'product2', 3),
    ('supplier2', 'product3', 3),
    ('supplier2', 'product4', 4))
AS Products(supplier_id, product_id, rating)
GROUP BY ROLLUP (supplier_id, rating)
```

CUBE

CUBE 是一种特定通用类型 Grouping Sets 的简写。代表着指定列表以及所有可能的子集和幂集。

例如：下面两个查询是等效的。

```
SELECT supplier_id, rating, product_id, COUNT(*)
FROM (VALUES
    ('supplier1', 'product1', 4),
    ('supplier1', 'product2', 3),
    ('supplier2', 'product3', 3),
    ('supplier2', 'product4', 4))
AS Products(supplier_id, product_id, rating)
GROUP BY CUBE (supplier_id, rating, product_id)

SELECT supplier_id, rating, product_id, COUNT(*)
FROM (VALUES
    ('supplier1', 'product1', 4),
    ('supplier1', 'product2', 3),
    ('supplier2', 'product3', 3),
    ('supplier2', 'product4', 4))
AS Products(supplier_id, product_id, rating)
GROUP BY GROUPING SET (
    ( supplier_id, product_id, rating ),
    ( supplier_id, product_id         ),
    ( supplier_id,             rating ),
    ( supplier_id                     ),
    (              product_id, rating ),
    (              product_id         ),
    (                          rating ),
    (                                 )
)
```

HAVING

HAVING 会删除 group 后不符合条件的行。 HAVING 和 WHERE 的不同点：WHERE 在 GROUP BY 之前过滤单独的数据行。HAVING 过滤 GROUP BY 生成的数据行。HAVING 条件中的每一列引用必须是明确的 grouping 列，除非它出现在聚合函数中。

 版权所有 © 华为技术有限公司

即使没有 GROUP BY 子句，HAVING 的存在也会使查询变成一个分组查询。这与查询包含聚合函数但没有 GROUP BY 子句时的情况相同。查询认为所有被选中的行形成一个单一的组，并且 SELECT 列表和 HAVING 子句只能从聚合函数中引用列。如果 HAVING 条件为真，这样的查询将发出一条记录，如果不为真，则发出零条记录。

```
SELECT SUM(amount)
FROM Orders
GROUP BY users
HAVING SUM(amount) > 50
```

2.6.6 Over 聚合

OVER 聚合通过排序后的范围数据为每行输入计算出聚合值。和 GROUP BY 聚合不同， OVER 聚合不会把结果通过分组减少到一行，它会为每行输入增加一个聚合值。

更多介绍和使用请参考开源社区文档：**Over聚合**。

语法格式

```
SELECT
  agg_func(agg_col) OVER (
    [PARTITION BY col1[, col2, ...]]
    ORDER BY time_col
    range_definition),
  ...
FROM ...
```

注意事项

- 当前仅支持 PRECEDING (无界或有界) 到 CURRENT ROW 范围内的窗口、FOLLOWING 所描述的区间并未支持。
- ORDER BY 必须指定于单个的时间属性。
- 可以在一个 SELECT 子句中定义多个 OVER 窗口聚合。然而，对于流式查询，由于目前的限制，所有聚合的 OVER 窗口必须是相同的。
- OVER 窗口需要数据是有序的。因为表没有固定的排序，所以 ORDER BY 子句是强制的。对于流式查询，Flink 目前只支持 OVER 窗口定义在升序（asc）的时间属性上。其他的排序不支持。

语法说明

```
SELECT order_id, order_time, amount,
  SUM(amount) OVER w AS sum_amount,
  AVG(amount) OVER w AS avg_amount
FROM Orders
WINDOW w AS (
  PARTITION BY product
  ORDER BY order_time
  RANGE BETWEEN INTERVAL '1' HOUR PRECEDING AND CURRENT ROW)
```

- **ORDER BY**：OVER 窗口需要数据是有序的。因为表没有固定的排序，所以 ORDER BY 子句是强制的。对于流式查询，Flink 目前只支持 OVER 窗口定义在升序（asc）的时间属性上。其他的排序不支持。
- **PARTITION BY**：OVER 窗口可以定义在一个分区表上。PARTITION BY 子句代表着每行数据只在其所属的数据分区进行聚合。
- **范围（RANGE）定义**：范围（RANGE）定义指定了聚合中包含了多少行数据。范围通过 BETWEEN 子句定义上下边界，其内的所有行都会聚合。Flink 只支持 CURRENT ROW 作为上边界。有两种方法可以定义范围：ROWS 间隔 和 RANGE 间隔：

版权所有 © 华为技术有限公司

a. **RANGE 间隔**

RANGE 间隔是定义在排序列值上的，在 Flink 里，排序列总是一个时间属性。下面的 RANG 间隔定义了聚合会在比当前行的时间属性小 30 分钟的所有行上进行。

```
RANGE BETWEEN INTERVAL '30' MINUTE PRECEDING AND CURRENT ROW
```

b. **ROW 间隔**

ROWS 间隔基于计数。它定义了聚合操作包含的精确行数。下面的 ROWS 间隔定义了当前行 + 之前的 10 行（也就是11行）都会被聚合。

```
ROWS BETWEEN 10 PRECEDING AND CURRENT ROW
```

- **WINDOW：** WINDOW 子句可用于在 SELECT 子句之外定义 OVER 窗口。它让查询可读性更好，也允许多个聚合共用一个窗口定义。

示例

查询为每个订单计算前一个小时之内接收到的同一产品所有订单的总金额。

```
SELECT order_id, order_time, amount,
  SUM(amount) OVER (
    PARTITION BY product
    ORDER BY order_time
    RANGE BETWEEN INTERVAL '1' HOUR PRECEDING AND CURRENT ROW
  ) AS one_hour_prod_amount_sum
FROM Orders
```

2.6.7 JOIN

Equi-join

语法格式

```
FROM tableExpression INNER | LEFT | RIGHT | FULL JOIN tableExpression
  ON value11 = value21 [ AND value12 = value22]
```

注意事项

- 目前仅支持 equi-join ，即 join 的联合条件至少拥有一个相等谓词。不支持任何 cross join 和 theta join。
- Join 的顺序没有进行优化，join 会按照 FROM 中所定义的顺序依次执行。请确保 join 所指定的表在顺序执行中不会产生不支持的 cross join （笛卡儿积）以致查询失败。
- 流查询中可能会因为不同行的输入数量导致计算结果的状态无限增长。请提供具有有效保留间隔的查询配置，以防止出现过多的状态。

示例

```
SELECT *
FROM Orders INNER JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders LEFT JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders RIGHT JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders FULL OUTER JOIN Product ON Orders.productId = Product.id;
```

 版权所有 © 华为技术有限公司

Time-windowed Join

功能描述

每条流的每一条数据会与另一条流上的不同时间区域的数据进行JOIN。

语法格式

```
from t1 JOIN t2 ON t1.key = t2.key AND TIMEBOUND_EXPRESSIO
```

语法描述

TIMEBOUND_EXPRESSION 有两种写法，如下：

- L.time between LowerBound(R.time) and UpperBound(R.time)
- R.time between LowerBound(L.time) and UpperBound(L.time)
- 带有时间属性(L.time/R.time)的比较表达式。

注意事项

时间窗口join需要至少一个 equi-join 谓词和一个限制了双方时间的 join 条件。

例如使用两个适当的范围谓词（<, <=, >=, >），一个 BETWEEN 谓词或一个比较两个输入表中相同类型的时间属性（即处理时间和事件时间）的相等谓词

比如，以下谓词是合法的窗口 join 条件：

- ltime = rtime
- ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
- ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND

示例

所有在收到后四小时内发货的 order 会与它们相关的 shipment 进行 join。

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.orderId AND
    o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime;
```

Expanding arrays into a relation

注意事项

目前尚未支持非嵌套的 WITH ORDINALITY 。

示例

```
SELECT users, tag
FROM Orders CROSS JOIN UNNEST(tags) AS t (tag);
```

Join 表函数(UDTF)

功能描述

将表与表函数的结果进行 join 操作。左表（outer）中的每一行将会与调用表函数所产生的所有结果中相关联行进行 join 。

注意事项

针对横向表的左外部连接当前仅支持文本常量 TRUE 作为谓词。

示例

如果表函数返回了空结果，左表（outer）的行将会被删除

```
SELECT users, tag
FROM Orders, LATERAL TABLE(unnest_udtf(tags)) t AS tag;
```

如果表函数返回了空结果，将会保留相对应的外部行并用空值填充

```
SELECT users, tag
FROM Orders LEFT JOIN LATERAL TABLE(unnest_udtf(tags)) t AS tag ON TRUE;
```

Join Temporal Table Function

功能描述

注意事项

目前仅支持在 Temporal Tables 上的 inner join

示例

假如Rates是一个 Temporal Table Function， join 可以使用 SQL 进行如下的表达:

```
SELECT
  o_amount, r_rate
FROM
  Orders,
  LATERAL TABLE (Rates(o_proctime))
WHERE
  r_currency = o_currency;
```

Join Temporal Tables

功能描述

与Temporal表进行join操作

语法格式

```
SELECT column-names
FROM table1  [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF table1.proctime [AS <alias2>]
ON table1.column-name1 = table2.key-name1
```

语法说明

- table1.proctime表示table1的proctime处理时间属性(计算列)
- 使用FOR SYSTEM_TIME AS OF table1.proctime表示当左边表的记录与右边的维表join时，只匹配当前处理时间维表所对应的的快照数据。

注意事项

仅支持带有处理时间的 temporal tables 的 inner 和 left join

示例

假设 LatestRates 是一个根据最新的 rates 物化的Temporal Table。

```
SELECT
  o.amout, o.currency, r.rate, o.amount * r.rate
FROM
```

 版权所有 © 华为技术有限公司

```
Orders AS o
JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
ON r.currency = o.currency;
```

2.6.8 OrderBy & Limit

OrderBy

功能描述

主要根据时间属性按照升序进行排序

注意事项

目前仅支持根据时间属性进行排序

示例

对订单根据订单时间进行升序排序

```
SELECT *
FROM Orders
ORDER BY orderTime;
```

Limit

功能描述

限制返回的数据结果个数

注意事项

LIMIT 查询需要有一个 ORDER BY 字句

示例

```
SELECT *
FROM Orders
ORDER BY orderTime
LIMIT 3;
```

2.6.9 Top-N

功能描述

Top-N 查询是根据列排序找到N个最大或最小的值。最大值集和最小值集都被视为是一种 Top-N 的查询。如果在批处理或流处理的表中需要显示出满足条件的 N 个最底层记录或最顶层记录， Top-N 查询将会十分有用。

语法格式

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     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]
```

 版权所有 © 华为技术有限公司

语法说明

- ROW_NUMBER(): 根据当前分区内的各行的顺序从第一行开始，依次为每一行分配一个唯一且连续的号码。目前，我们只支持 ROW_NUMBER 在 over 窗口函数中使用。未来将会支持 RANK() 和 DENSE_RANK()函数。
- PARTITION BY col1[, col2...]: 指定分区列，每个分区都将会有一个 Top-N 结果。
- ORDER BY col1 [asc|desc][, col2 [asc|desc]...]: 指定排序列，不同列的排序方向可以不一样。
- WHERE rownum <= N: Flink 需要 rownum <= N 才能识别一个查询是否为 Top-N 查询。 其中， N 代表最大或最小的 N 条记录会被保留。
- [AND conditions]: 在 where 语句中，可以随意添加其他的查询条件，但其他条件只允许通过 AND 与 rownum <= N 结合使用。

注意事项

- TopN 查询的结果会带有更新。
- Flink SQL 会根据排序键对输入的流进行排序。
- 如果 top N 的记录发生了变化，变化的部分会以撤销、更新记录的形式发送到下游。
- 如果 top N 记录需要存储到外部存储，则结果表需要拥有相同与 Top-N 查询相同的唯一键。

示例

查询每个分类实时销量最大的五个产品

```
SELECT *
  FROM (
     SELECT *,
         ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) as row_num
     FROM ShopSales)
  WHERE row_num <= 5;
```

2.6.10 去重

功能描述

对在列的集合内重复的行进行删除，只保留第一行或最后一行数据。

语法格式

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]
       ORDER BY time_attr [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1
```

语法说明

- ROW_NUMBER(): 从第一行开始，依次为每一行分配一个唯一且连续的号码。
- PARTITION BY col1[, col2...]: 指定分区的列，例如去重的键。

- ORDER BY time_attr [asc|desc]: 指定排序的列。所指定的列必须为时间属性。目前仅支持proctime。升序（ ASC ）排列指只保留第一行，而降序排列（ DESC ）则只保留最后一行。
- WHERE rownum = 1: Flink 需要 rownum = 1 以确定该查询是否为去重查询。

注意事项

无

示例

根据order_id对数据进行去重，其中proctime为事件时间属性列

```
SELECT order_id, user, product, number
  FROM (
    SELECT *,
        ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY proctime ASC) as row_num
    FROM Orders)
  WHERE row_num = 1;
```

2.7 函数

2.7.1 自定义函数

概述

DLI支持三种自定义函数：

- UDF：自定义函数，支持一个或多个输入参数，返回一个结果值。
- UDTF：自定义表值函数，支持一个或多个输入参数，可返回多行多列。
- UDAF：自定义聚合函数，将多条记录聚合成一个值。

说明

暂不支持通过python写UDF、UDTF、UDAF自定义函数。

POM 依赖

```
<dependency>
   <groupId>org.apache.flink</groupId>
   <artifactId>flink-table-common</artifactId>
   <version>1.15.0</version>
   <scope>provided</scope>
</dependency>
```

使用方式

1. 将写好的自定义函数打成JAR包，并上传到OBS上。
2. 在DLI管理控制台的左侧导航栏中，单击数据管理>“程序包管理”，然后单击创建，并使用OBS中的jar包创建相应的程序包。
3. 在DLI管理控制台的左侧导航栏中，单击作业管理>“Flink作业”，在需要编辑作业对应的“操作”列中，单击“编辑”，进入作业编辑页面。
4. 在“运行参数设置”页签，“UDF Jar”选择创建的程序包，单击“保存”。

 版权所有 © 华为技术有限公司

5. 选定JAR包以后，SQL里添加UDF声明语句，就可以像普通函数一样使用了。
```
CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
```

UDF

UDF函数需继承ScalarFunction函数，并实现eval方法。open函数及close函数可选。

编写代码示例

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;

public class UdfScalarFunction extends ScalarFunction {
    private int factor = 12;
    public UdfScalarFunction() {
        this.factor = 12;
    }
    /**
     * 初始化操作，可选
     * @param context
     */
    @Override
    public void open(FunctionContext context) {}
    /**
     * 自定义逻辑
     * @param s
     * @return
     */
    public int eval(String s) {
        return s.hashCode() * factor;
    }
    /**
     * 可选
     */
    @Override   public void close() {}
}
```

使用示例

```
CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
INSERT INTO sink_stream select udf_test(attr) FROM source_stream;
```

UDTF

UDTF函数需继承TableFunction函数，并实现eval方法。open函数及close函数可选。如果需要UDTF返回多列，只需要将返回值声明成Tuple或Row即可。如果使用Row，需要重载getResultType声明返回的字段类型。

编写代码示例

```
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;

public class UdfTableFunction extends TableFunction<Row> {
    /**
     * 初始化操作，可选
     * @param context
     */
    @Override
    public void open(FunctionContext context) {}
    public void eval(String str, String split) {
        for (String s : str.split(split)) {
            Row row = new Row(2);
```

```
            row.setField(0, s);
            row.setField(1, s.length());
            collect(row);
        }
    }
    /**
     * 函数返回类型声明
     * @return
     */
    @Override
    public TypeInformation<Row> getResultType() {
        return Types.ROW(Types.STRING, Types.INT);
    }
    /**
     * 可选
     */
    @Override
    public void close() {}
}
```

使用示例

UDTF支持CROSS JOIN和LEFT JOIN，在使用UDTF时需要带上 LATERAL 和TABLE 两个关键字。

- CROSS JOIN：对于左表的每一行数据，假设UDTF不产生输出，则这一行不进行输出。
- LEFT JOIN：对于左表的每一行数据，假设UDTF不产生输出，这一行仍会输出，UDTF相关字段用null填充。

```
CREATE FUNCTION udtf_test AS 'com.huaweicompany.udf.TableFunction';
// CROSS JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length);
// LEFT JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

UDAF

UDAF函数需继承AggregateFunction函数。首先需要创建一个用来存储计算结果的Accumulator，如示例里的WeightedAvgAccum。

编写代码示例

```
public class WeightedAvgAccum {
public long sum = 0;
public int count = 0;
}
```

```
import org.apache.flink.table.functions.AggregateFunction;

import java.util.Iterator;

/**
 * 第一个类型变量为聚合函数返回的类型，第二个类型变量为Accumulator类型
 * Weighted Average user-defined aggregate function.
 */
public class UdfAggFunction extends AggregateFunction<Long, WeightedAvgAccum> {
    // 初始化Accumulator
    @Override
    public WeightedAvgAccum createAccumulator() {
        return new WeightedAvgAccum();
    }
    // 返回Accumulator存储的中间计算值
    @Override
```

```
    public Long getValue(WeightedAvgAccum acc) {
        if (acc.count == 0) {
            return null;
        } else {
            return acc.sum / acc.count;
        }
    }
    // 根据输入更新中间计算值
    public void accumulate(WeightedAvgAccum acc, long iValue) {
        acc.sum += iValue;
        acc.count += 1;
    }
    // Restract撤回操作，和accumulate操作相反
    public void retract(WeightedAvgAccum acc, long iValue) {
        acc.sum -= iValue;
        acc.count -= 1;
    }
    // 合并多个accumulator值
    public void merge(WeightedAvgAccum acc, Iterable<WeightedAvgAccum> it) {
        Iterator<WeightedAvgAccum> iter = it.iterator();
        while (iter.hasNext()) {
            WeightedAvgAccum a = iter.next();
            acc.count += a.count;
            acc.sum += a.sum;
        }
    }
    // 重置中间计算值
    public void resetAccumulator(WeightedAvgAccum acc) {
        acc.count = 0;
        acc.sum = 0L;
    }
}
```

使用示例

```
CREATE FUNCTION udaf_test AS 'com.huaweicompany.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

2.7.2 自定义函数类型推导

操作场景

类型推导包含了验证输入值、派生参数和返回值数据类型。从逻辑角度看，Planner需要知道数据类型、精度和小数位数；从 JVM 角度来看，Planner 在调用自定义函数时需要知道如何将内部数据结构表示为JVM对象。

Flink 自定义函数实现了自动的类型推导提取，通过反射从函数的类及其求值方法中派生数据类型。然而以反射方式提取数据类型并不总是成功的，比如UDTF中常见的Row类型。

由于 Flink 1.11 起引入了新的自定义函数注册接口，使用了新的自定义函数类型推断机制，因此原先1.10 重载 getResultType 声明返回字段类型的方式将不再可用。继续使用会抛出如下异常：

```
Caused by: org.apache.flink.table.api.ValidationException: Cannot extract a data type from a pure 'org.apache.flink.types.Row' class. Please use annotations to define field names and field types.
```

目前 Flink 1.15 可以通过使用DataTypeHint 和FunctionHint 注解相关参数、类或方法来支持提取过程。

代码示例

Table（类似于 SQL 标准）是一种强类型的 API，函数的参数和返回类型都必须映射到 Table API 的数据类型，参见**Table API数据类型**。

版权所有 © 华为技术有限公司

如果需要更高级的类型推导逻辑，您可以在每个自定义函数中显式重写 getTypeInference( ) 方法。

建议使用注解方式，因为它可使自定义类型推导逻辑保持在受影响位置附近，而在其他位置则保持默认状态。

```
importorg.apache.flink.table.annotation.DataTypeHint;
importorg.apache.flink.table.annotation.FunctionHint;
importorg.apache.flink.table.functions.FunctionContext;
importorg.apache.flink.table.functions.TableFunction;
importorg.apache.flink.types.Row;
publicclassUdfTableFunctionextendsTableFunction<Row>{
    /**
     *初始化操作，可选
     *@paramcontext
     */
    @Override
     public void open(FunctionContextcontext) {  }

    @FunctionHint(output=@DataTypeHint("ROW<s STRING, i INT>"))
    publicvoideval(String str, String split) {
        for (String s: str.split(split)) {
            Row row=new Row(2);
            row.setField(0, s);
            row.setField(1, s.length());
            collect(row);
        }
    }
    /**
    * 可选
    */
   @Override
   public void close() {}
}
```

使用示例

UDTF支持CROSS JOIN和LEFT JOIN，在使用UDTF时需要带上 LATERAL 和TABLE 两个关键字。

- CROSS JOIN：对于左表的每一行数据，假设UDTF不产生输出，则这一行不进行输出。
- LEFT JOIN：对于左表的每一行数据，假设UDTF不产生输出，这一行仍会输出，UDTF相关字段用null填充。

```
CREATE FUNCTION udtf_test AS 'com.huaweicompany.udf.TableFunction';-- CROSS JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length);-- LEFT JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN
LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

2.7.3 自定义函数参数传递

操作场景

如果您的自定义函数需要在多个作业中使用，但对于不同作业某些参数值不同，直接在UDF中修改较为复杂。您可以在Flink OpenSource SQL编辑页面，自定义配置中配置参数**pipeline.global-job-parameters**，在UDF代码中获取该参数并使用。如需修改参数值，直接在FlinkOpenSource SQL编辑页面，自定义配置中修改该参数值，即可达到快速修改UDF参数值的目的。

 版权所有 © 华为技术有限公司

操作步骤

自定义函数中提供了可选的open(FunctionContext context)方法，FunctionContext具备参数传递功能，自定义配置项通过此对象来传递。自定义函数的参数传递操作步骤如下：

1. 在Flink OpenSource SQL编辑页面右侧自定义配置中添加参数pipeline.global-job-parameters，格式如下：
   ```
   pipeline.global-job-parameters=k1:v1,"k2:v1,v2",k3:"str:ing","k4:str""ing"
   ```
   该配置定义了如表2-77的map。

   表 2-77 pipeline.global-job-parameters 示例

key	value
k1	v1
k2	v1,v2
k3	str:ing
k4	str""ing

   说明

   - FunctionContext#getJobParameter只能获取pipeline.global-job-parameters这一配置项的值。因此需要将UDF用到的所有配置项全部写入到pipeline.global-job-parameters中。
   - key和value之间通过冒号(:)分隔，所有key-value用逗号(,)连接。
   - 如果key或value中含有逗号(,)，则需要用双引号(")将key:value整个包围起来。参考k2。
   - 如果key或value中含有半角冒号(:)，则需要用双引号(")将key或value包围起来。参考k3。
   - 如果key或value中含有双引号(")，则需要通过连写两个双引号("")进行转义，也需要用双引号(")将key:value整个包围起来。参考k4。
2. 在自定义函数代码中，通过FunctionContext#getJobParameter获取map的各项内容，代码示例如下：
   ```
   context.getJobParameter("url","jdbc:mysql://xx.xx.xx.xx:3306/table");
   context.getJobParameter("driver","com.mysql.jdbc.Driver");
   context.getJobParameter("user","user");
   context.getJobParameter("password","password");
   ```

代码示例

以下是一个UDF示例：通过pipeline.global-job-parameters传入连接数据库需要的url、user、password等参数，获取udf_info表数据后和流数据拼接成json输出。

表 2-78 udf_info 表

key	value
class	class-4

版权所有 © 华为技术有限公司

SimpleJsonBuild.java

```
package udf;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.HashMap;
import java.util.Map;

public class SimpleJsonBuild extends ScalarFunction {
  private static final Logger LOG = LoggerFactory.getLogger(SimpleJsonBuild.class);
  String remainedKey;
  String remainedValue;

  private Connection initConnection(Map<String, String> userParasMap) {
      String url = userParasMap.get("url");
      String driver = userParasMap.get("driver");
      String user = userParasMap.get("user");
      String password = userParasMap.get("password");
      Connection conn = null;
      try {
          Class.forName(driver);
          conn = DriverManager.getConnection(url, user, password);
          LOG.info("connect successfully");
      } catch (Exception e) {
          LOG.error(String.valueOf(e));
      }
      return conn;
  }

  @Override
  public void open(FunctionContext context) throws Exception {
      Map<String, String> userParasMap = new HashMap<>();
      Connection connection;
      PreparedStatement pstmt;
      ResultSet rs;

      String url = context.getJobParameter("url","jdbc:mysql://xx.xx.xx.xx:3306/table");
      String driver = context.getJobParameter("driver","com.mysql.jdbc.Driver");
      String user = context.getJobParameter("user","user");
      String password = context.getJobParameter("password","password");

      userParasMap.put("url", url);
      userParasMap.put("driver", driver);
      userParasMap.put("user", user);
      userParasMap.put("password", password);

      connection = initConnection(userParasMap);
      String sql = "select `key`, `value` from udf_info";
      pstmt = connection.prepareStatement(sql);
      rs = pstmt.executeQuery();

      while (rs.next()) {
          remainedKey = rs.getString(1);
          remainedValue = rs.getString(2);
      }
  }

  public String eval(String... params) throws IOException {
      if (params != null && params.length != 0 && params.length % 2 <= 0) {
```

```
        HashMap<String, String> hashMap = new HashMap();
        for (int i = 0; i < params.length; i += 2) {
            hashMap.put(params[i], params[i + 1]);
            LOG.debug("now the key is " + params[i].toString() + "; now the value is " + params[i +
1].toString());
        }
        hashMap.put(remainedKey, remainedValue);
        ObjectMapper mapper = new ObjectMapper();
        String result = "{}";
        try {
            result = mapper.writeValueAsString(hashMap);
        } catch (Exception ex) {
            LOG.error("Get result failed." + ex.getMessage());
        }
        LOG.debug(result);
        return result;
    } else {
        return "{}";
    }
 }

  public static void main(String[] args) throws IOException {
     SimpleJsonBuild  sjb = new SimpleJsonBuild();
     System.out.println(sjb.eval("json1", "json2", "json3", "json4"));
 }
}
```

在Flink OpenSource SQL编辑页面右侧**自定义配置**中添加参数pipeline.global-job-parameters

```
pipeline.global-job-parameters=url:'jdbc:mysql://x.x.x.x:xxxx/
test',driver:com.mysql.jdbc.Driver,user:xxx,password:xxx
```

Flink OpenSource SQL

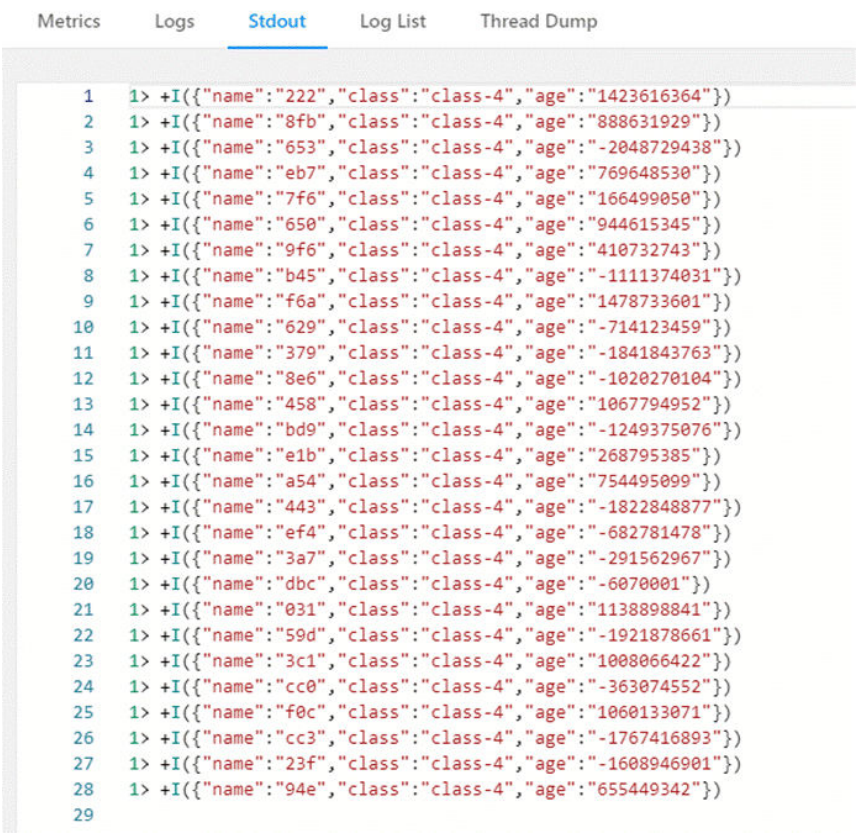
```
create function SimpleJsonBuild AS 'udf.SimpleJsonBuild';
create table dataGenSource(user_id string, amount int) with (
 'connector' = 'datagen',
 'rows-per-second' = '1', --每秒生成一条数据
 'fields.user_id.kind' = 'random', --为字段user_id指定random生成器
 'fields.user_id.length' = '3' --限制user_id长度为3
);
create table printSink(message STRING) with ('connector' = 'print');
insert into
printSink
SELECT
SimpleJsonBuild("name", user_id, "age", cast(amount as string))
from
dataGenSource;
```

运行结果

单击Flink作业操作列下的“更多 > FlinkUI > Task Managers > Stdout”查看输出结果:

 版权所有 © 华为技术有限公司

图 2-6 Flink 作业输出结果



2.7.4 内置函数

具体使用请参考开源社区文档：**内置函数**。

2.7.4.1 比较函数

表 2-79 比较函数

SQL函数	返回类型	描述
value1 = value2	BOOLEAN	如果 value1 等于 value2 返回 TRUE； 如果 value1 或者 value2 为 NULL 返回 UNKNOWN。
value1 <> value2	BOOLEAN	如果 value1 不等于 value2 返回 TRUE； 如果 value1 或 value2 为 NULL 返回 UNKNOWN。
value1 > value2	BOOLEAN	如果 value1 大于 value2 返回 TRUE； 如果 value1 或 value2 为 NULL 返回 UNKNOWN。

版权所有 © 华为技术有限公司

SQL函数	返回类型	描述
value1 >= value2	BOOLEAN	如果 value1 大于或等于 value2 返回 TRUE； 如果 value1 或 value2 为 NULL 返回 UNKNOWN。
value1 < value2	BOOLEAN	如果 value1 小于 value2 返回 TRUE； 如果 value1 或 value2 为 NULL 返回 UNKNOWN。
value1 <= value2	BOOLEAN	如果 value1 小于或等于 value2 返回 TRUE； 如果 value1 或 value2 为 NULL 返回 UNKNOWN。
value IS NULL	BOOLEAN	如果值为 NULL 返回 TRUE。
value IS NOT NULL	BOOLEAN	如果值不为 NULL 返回 TRUE。
value1 IS DISTINCT FROM value2	BOOLEAN	value1和value2的数据类型和值不完全相同返回 TRUE。 value1和value2的数据类型和值都相同返回 FALSE。 将 NULL 视为相同。 例如： 1 IS DISTINCT FROM NULL 返回 TRUE； NULL IS DISTINCT FROM NULL 返回 FALSE。
value1 IS NOT DISTINCT FROM value2	BOOLEAN	value1和value2的数据类型和值都相同返回 TRUE。 value1和value2的数据类型和值不完全相同则返回 FALSE。 将 NULL 视为相同。 例如： 1 IS NOT DISTINCT FROM NULL 返回 FALSE； NULL IS NOT DISTINCT FROM NULL 返回 TRUE。

版权所有 © 华为技术有限公司

SQL函数	返回类型	描述
value1 BETWEEN [ASYMMETRIC | SYMMETRIC] value2 AND value3	BOOLEAN	默认或使用 ASYMMETRIC 关键字的情况下，如果 value1 大于等于 value2 且小于等于 value3 返回 TRUE。 使用 SYMMETRIC 关键字则 value1 在 value2 和 value3 之间返回 TRUE。 当 value2 或 value3 为 NULL 时，返回 FALSE 或 UNKNOWN。 例如： ● 12 BETWEEN 15 AND 12 返回 FALSE； ● 12 BETWEEN SYMMETRIC 15 AND 12 返回 TRUE； ● 12 BETWEEN 10 AND NULL 返回 UNKNOWN； ● 12 BETWEEN NULL AND 10 返回 FALSE； ● 12 BETWEEN SYMMETRIC NULL AND 12 返回 UNKNOWN。
value1 NOT BETWEEN [ASYMMETRIC | SYMMETRIC] value2 AND value3	BOOLEAN	默认或使用 ASYMMETRIC 关键字的情况下，如果 value1 小于 value2 或大于 value3，则返回 TRUE。 使用 SYMMETRIC 关键字则 value1 不在 value2 和 value3 之间返回 TRUE。 当 value2 或 value3 为 NULL 时，返回 TRUE 或 UNKNOWN。 例如： ● 12 NOT BETWEEN 15 AND 12 返回 TRUE； ● 12 NOT BETWEEN SYMMETRIC 15 AND 12 返回 FALSE； ● 12 NOT BETWEEN NULL AND 15 返回 UNKNOWN； ● 12 NOT BETWEEN 15 AND NULL 返回 TRUE； ● 12 NOT BETWEEN SYMMETRIC 12 AND NULL 返回 UNKNOWN。
string1 LIKE string2 [ESCAPE char]	BOOLEAN	如果 string1 匹配 string2 返回 TRUE； 如果 string1 或 string2 为 NULL 返回 UNKNOWN。 如果需要可以定义转义字符。尚不支持转义字符。

版权所有 © 华为技术有限公司

SQL函数	返回类型	描述
string1 NOT LIKE string2 [ESCAPE char]	BOOLEAN	如果 string1 与 string2 不匹配返回 TRUE； 如果 string1 或 string2 为 NULL 返回 UNKNOWN。 如果需要可以定义转义字符。尚不支持转义字符。
string1 SIMILAR TO string2 [ESCAPE char]	BOOLEAN	如果 string1 匹配 SQL 正则表达式 string2 返回 TRUE； 如果 string1 或 string2 为 NULL 返回 UNKNOWN。 如果需要可以定义转义字符。尚不支持转义字符。
string1 NOT SIMILAR TO string2 [ESCAPE char]	BOOLEAN	如果 string1 与 SQL 正则表达式 string2 不匹配返回 TRUE； 如果 string1 或 string2 为 NULL 返回 UNKNOWN。 如果需要可以定义转义字符。尚不支持转义字符。
value1 IN (value2 [, value3]*)	BOOLEAN	在给定列表 (value2, value3, …) 中存在value1返回TRUE。 当列表包含NULL，如果可以找到value1则返回TRUE，否则返回UNKNOWN。 如果value1为NULL 则始终返回UNKNOWN。 例如： ● 4 IN (1, 2, 3) 返回 FALSE； ● 1 IN (1, 2, NULL) 返回 TRUE； ● 4 IN (1, 2, NULL) 返回 UNKNOWN。
value1 NOT IN (value2 [, value3]*)	BOOLEAN	在给定列表 (value2, value3, …) 中不存在 value1 返回 TRUE。 当列表包含NULL，如果可以找到value1则 返回FALSE，否则返回UNKNOWN。 如果value1为NULL，则始终返回UNKNOWN。 例如： ● 4 NOT IN (1, 2, 3) 返回 TRUE； ● 1 NOT IN (1, 2, NULL) 返回 FALSE； ● 4 NOT IN (1, 2, NULL) 返回 UNKNOWN。
EXISTS (sub-query)	BOOLEAN	如果子查询至少返回一行则返回 TRUE。 仅支持可以在 join 和分组操作中可以被重写的操作。对于流式查询，该操作在 join 和分组操作中被重写。根据输入行的数量计算查询结果所需的状态可能会无限增长。 请提供具有有效保留间隔的查询配置，以防止状态过大。

版权所有 © 华为技术有限公司

SQL函数	返回类型	描述
value IN (sub-query)	BOOLEAN	如果 value 等于子查询结果集中的一行则返回 TRUE。
value NOT IN (sub-query)	BOOLEAN	如果 value 不包含于子查询返回的行则返回 TRUE。

2.7.4.2 逻辑函数

表 2-80 逻辑函数

SQL函数	返回类型	描述
boolean1 OR boolean2	BOOLEAN	如果 boolean1 为 TRUE 或 boolean2 为 TRUE 返回 TRUE。支持三值逻辑。 例如 true || Null(BOOLEAN) 返回 TRUE。
boolean1 AND boolean2	BOOLEAN	如果 boolean1 和 boolean2 都为 TRUE 返回 TRUE。支持三值逻辑。 例如 true && Null(BOOLEAN) 返回 UNKNOWN。
NOT boolean	BOOLEAN	如果布尔值为 FALSE 返回 TRUE；如果布尔值为 TRUE 返回 FALSE；如果布尔值为 UNKNOWN 返回 UNKNOWN。
boolean IS FALSE	BOOLEAN	如果布尔值为 FALSE 返回 TRUE；如果 boolean 为 TRUE 或 UNKNOWN 返回 FALSE。
boolean IS NOT FALSE	BOOLEAN	如果 boolean 为 TRUE 或 UNKNOWN 返回 TRUE；如果 boolean 为 FALSE 返回 FALSE。
boolean IS TRUE	BOOLEAN	如果 boolean 为 TRUE 返回 TRUE；如果 boolean 为 FALSE 或 UNKNOWN 返回 FALSE。
boolean IS NOT TRUE	BOOLEAN	如果 boolean 为 FALSE 或 UNKNOWN 返回 TRUE；如果布尔值为 TRUE 返回 FALSE。
boolean IS UNKNOWN	BOOLEAN	如果布尔值为 UNKNOWN 返回 TRUE；如果 boolean 为 TRUE 或 FALSE 返回 FALSE。
boolean IS NOT UNKNOWN	BOOLEAN	如果 boolean 为 TRUE 或 FALSE 返回 TRUE；如果布尔值为 UNKNOWN 返回 FALSE。

版权所有 © 华为技术有限公司

2.7.4.3 算术函数

表 2-81 算术函数

运算符	描述
+ numeric	返回 numeric。
- numeric	返回 numeric 的相反数。
numeric1 + numeric2	返回 numeric1 加 numeric2
numeric1 - numeric2	返回 numeric1 减 numeric2。
numeric1 * numberic2	返回 numeric1 乘以 numeric2。
numeric1 / numeric2	返回 numeric1 除以 numeric2。
numeric1 % numeric2	返回 numeric1 除以 numeric2 的余数（模数）。仅当 numeric1 为负时，结果才为负。
POWER(numeric1, numeric2)	返回 numeric1 的 numeric2 次方。
ABS(numeric)	返回 numeric 的绝对值。
SQRT(numeric)	返回 numeric 的平方根。
LN(numeric)	返回 numeric 的自然对数（以 e 为底）。
LOG10(numeric)	返回以 10 为底的 numeric 的对数。
LOG2(numeric)	返回以 2 为底的 numeric 的对数。
LOG(numeric2) LOG(numeric1, numeric2)	当用一个参数调用时，返回 numeric2 的自然对数。当使用两个参数调用时，此函数返回 numeric2 以 numeric1 为底的对数。numeric2 必须大于 0，numeric1 必须大于 1。
EXP(numeric)	返回 e 的 numeric 次幂。
CEIL(numeric) CEILING(numeric)	向上取整，并返回大于或等于 numeric 的最小整数。
FLOOR(numeric)	向下取整，并返回小于或等于 numeric 的最大整数。
SIN(numeric)	返回 numeric 的正弦值。
SINH(numeric)	返回 numeric 的双曲正弦值。返回类型为 DOUBLE。
COS(numeric)	返回 numeric 的正切值。

版权所有 © 华为技术有限公司

运算符	描述
TAN(numeric)	计算给定A的正切值。
TANH(numeric)	返回 numeric 的双曲正切值。返回类型为 DOUBLE。
COT(numeric)	返回 numeric 的余切值。
ASIN(numeric)	返回 numeric 的反正弦值。
ACOS(numeric)	返回 numeric 的反余弦值。
ATAN(numeric)	返回 numeric 的反正切值。
ATAN2(numeric1, numeric2)	返回坐标 (numeric1, numeric2) 的反正切。
COSH(numeric)	返回 numeric 的双曲余弦值。返回值类型为 DOUBLE。
DEGREES(numeric)	返回弧度 numeric 的度数表示。
RADIANS(numeric)	返回度数 numeric 的弧度表示。
SIGN(numeric)	返回 numeric 的符号。
ROUND(numeric, INT)	返回 numeric 四舍五入保留 INT 小数位的值。
PI()	返回无比接近 pi 的值。
E()	返回无比接近 e 的值。
RAND()	返回 [0.0, 1.0) 范围内的伪随机双精度值。
RAND(INT)	返回范围为 [0.0, 1.0) 的伪随机双精度值，初始种子为 INT。 如果两个 RAND 函数具有相同的初始种子，它们将返回相同的数字序列。
RAND_INTEGER(INT)	返回 [0, INT) 范围内的伪随机整数。
RAND_INTEGER(INT1, INT2)	返回范围为 [0, INT2) 的伪随机整数，初始种子为 INT1。 如果两个 RAND_INTGER 函数具有相同的初始种子和边界，它们将返回相同的数字序列。
UUID()	根据 RFC 4122 类型 4（伪随机生成）UUID，返回 UUID（通用唯一标识符）字符串。 例如“3d3c68f7-f608-473f-b60c-b0c44ad4cc4e”，UUID 是使用加密强的伪随机数生成器生成的。
BIN(INT)	以二进制格式返回 INTEGER 的字符串表示形式。如果 INTEGER 为 NULL，则返回 NULL。 例如 4.bin() 返回“100”，12.bin() 返回“1100”。

版权所有 © 华为技术有限公司

运算符	描述
HEX(numeric) HEX(string)	以十六进制格式返回整数 numeric 值或 STRING 的字符串表示形式。如果参数为 NULL，则返回 NULL。 例如数字 20 返回“14”，数字 100 返回“64”，字符串“hello,world” 返回“68656C6C6F2C776F726C64”。
TRUNCATE(numeric1, integer2)	返回截取 integer2 位小数的数字。如果 numeric1 或 integer2 为 NULL，则返回 NULL。 如果 integer2 为 0，则结果没有小数点或小数部分。integer2 可以为负数，使值的小数点左边的 integer2 位变为零。 此函数也可以传入只有一个 numeric1 参数且不设置 Integer2 以使用。 如果未设置 Integer2 则 Integer2 为 0。 例如 42.324.truncate(2) 为 42.32，42.324.truncate() 为 42.0。

2.7.4.4 字符串函数

字符串函数

数据湖探索（DLI）提供了丰富的字符串函数，用于处理和转换字符串数据。这些函数包括拼接、大小写转换、截取子串、替换、正则匹配、编码解码、格式转换等。此外，还支持字符串长度计算、位置查找、填充、反转等功能，以及从JSON字符串中提取值的JSON_VAL函数。这些功能广泛应用于数据清洗、文本处理和数据分析场景，为开发者提供强大的工具支持。

字符串函数简介请参考**表2-82**，更多内容参考**Apache Flink**。

表 2-82 字符串函数

SQL函数	描述
string1 || string2	返回 STRING1 和 STRING2 的连接。
CHAR_LENGTH(string) CHARACTER_LENGTH(string)	返回字符串中的字符数。
UPPER(string)	以大写形式返回字符串。
LOWER(string)	以小写形式返回字符串。
POSITION(string1 IN string2)	返回 STRING2 中第一次出现 STRING1 的位置（从 1 开始）； 如果在 STRING2 中找不到 STRING1 返回 0。
TRIM([BOTH | LEADING | TRAILING] string1 FROM string2)	返回从 STRING1 中删除以字符串 STRING2 开头/结尾/开头且结尾的字符串的结果。默认情况下，两边的空格都会被删除。

版权所有 © 华为技术有限公司

SQL函数	描述
LTRIM(string)	返回从 STRING 中删除左边空格的字符串。 例如 ' This is a test String.'.ltrim() 返回 'This is a test String.'。
RTRIM(string)	返回从 STRING 中删除右边空格的字符串。 例如 'This is a test String. '.ltrim() 返回 'This is a test String.'。
REPEAT(string, int)	返回 INT 个 string 连接的字符串。 例如 REPEAT('This is a test String.', 2) 返回 "This is a test String.This is a test String."。
REGEXP_REPLACE(string1, string2, string3)	返回 STRING1 所有与正则表达式 STRING2 匹配的子字符串被 STRING3 替换后的字符串。 例如 'foobar'.regexpReplace('oo|ar', '') 返回 "fb"。
OVERLAY(string1 PLACING string2 FROM integer1 [FOR integer2])	返回一个字符串，该字符串从位置 INT1 用 STRING2 替换 STRING1 的 INT2（默认为 STRING2 的长度）字符。 例如 'xxxxxtest'.overlay('xxxx', 6) 返回 "xxxxxxxxx"；'xxxxxtest'.overlay('xxxx', 6, 2) 返回 "xxxxxxxxxst"。
SUBSTRING(string FROM integer1 [FOR integer2])	返回 STRING 从位置 INT1 开始，长度为 INT2（默认到结尾）的子字符串。
REPLACE(string1, string2, string3)	返回一个新字符串，它用 STRING1 中的 STRING3（非重叠）替换所有出现的 STRING2。 例如 'hello world'.replace('world', 'flink') 返回 'hello flink'； 'ababab'.replace('abab', 'z') 返回 'zab'。
REGEXP_EXTRACT(string1, string2[, integer])	将字符串 STRING1 按照 STRING2 正则表达式的规则拆分，返回指定 INTEGER1 处位置的字符串。 正则表达式匹配组索引从 1 开始， 0 表示匹配整个正则表达式。此外，正则表达式匹配组索引不应超过定义的组数。 例如 REGEXP_EXTRACT('foothebar', 'foo(.*?)(bar)', 2) 返回 "bar"。
INITCAP(string)	返回新形式的 STRING，其中每个单词的第一个字符转换为大写，其余字符转换为小写。这里的单词表示字母数字的字符序列。
CONCAT(string1, string2, ...)	返回连接 string1，string2， … 的字符串。如果有任一参数为 NULL，则返回 NULL。 例如 CONCAT('AA', 'BB', 'CC') 返回 "AABBCC"。

版权所有 © 华为技术有限公司

SQL函数	描述
CONCAT_WS(string1, string2, string3, ...)	返回将 STRING2， STRING3， … 与分隔符 STRING1 连接起来的字符串。 在要连接的字符串之间添加分隔符。 如果 STRING1 为 NULL，则返回 NULL。 与 concat() 相比，concat_ws() 会自动跳过 NULL 参数。 例如 concat_ws('~', 'AA', Null(STRING), 'BB', '', 'CC') 返回 "AA~BB~~CC".
LPAD(string1, integer, string2)	返回从 string1 靠左填充 string2 到 INT 长度的新字符串。 如果 string1 的长度小于 INT 值，则返回 string1 缩 短为整数字符。 例如 LPAD('hi', 4, '??') 返回 "??hi"；LPAD('hi', 1, '??') 返回 `“h”。
RPAD(string1, integer, string2)	返回从 string1 靠右边填充 string2 到 INT 长度的新字符串。 如果 string1 的长度小于 INT 值，则返回 string1 缩 短为长度为 INT 的新字符串。 例如 RPAD('hi', 4, '??') 返回 "hi??", RPAD('hi', 1, '??') 返回 "h"。
FROM_BASE64(string)	返回字符串 string1 的 base64 解码的结果；如果字符串为 NULL，则返回 NULL。 例如 FROM_BASE64('aGVsbG8gd29ybGQ=') 返回 "hello world"。
TO_BASE64(string)	返回字符串 string 的 base64 编码的结果；如果字符串为 NULL，则返回 NULL。 例如 TO_BASE64('hello world') 返回 "aGVsbG8gd29ybGQ="。
ASCII(string)	返回字符串 string 第一个字符的数值。如果字符串为 NULL 则返回 NULL。 例如 ascii('abc') 返回 97，ascii(CAST(NULL AS VARCHAR)) 返回 NULL。
CHR(integer)	返回二进制等于 integer 的 ASCII 字符。 如果整数 integer 大于 255，我们先将得到整数对 255 取模数， 并返回模数的 CHR。 如果整数为 NULL，则返回 NULL。 例如 chr(97) 返回 a，chr(353) 返回 a， ascii(CAST(NULL AS VARCHAR)) 返回 NULL。
DECODE(binary, string)	使用提供的字符集（‘US-ASCII’，‘ISO-8859-1’，‘UTF-8’，‘UTF-16BE’，‘UTF-16LE’，‘UTF-16’）解码。 如果任一参数为空，则结果也将为空。

版权所有 © 华为技术有限公司

SQL函数	描述
ENCODE(string1, string2)	使用提供的字符集（‘US-ASCII’，‘ISO-8859-1’，‘UTF-8’，‘UTF-16BE’，‘UTF-16LE’，‘UTF-16’）编码。 如果任一参数为空，则结果也将为空。
INSTR(string1, string2)	返回 string2 在 string1 中第一次出现的位置。如果有任一参数为 NULL，则返回 NULL。
LEFT(string, integer)	返回字符串中最左边的长度为 integer 值的字符串。如果 integer 为负，则返回 EMPTY 字符串。如果有任一参数 为 NULL 则返回 NULL。
RIGHT(string, integer)	返回字符串中最右边的长度为 integer 值的字符串。如果 integer 为负，则返回 EMPTY 字符串。如果有任一参数 为 NULL 则返回 NULL。
LOCATE(string1, string2[, integer])	返回 string2 中 string1 在位置 integer 之后第一次出现的位置。未找到返回 0。如果有任一参数为 NULL 则返回 NULL。
PARSE_URL(string1, string2[, string3])	从 URL 返回指定的部分。string2 的有效值包括“HOST”，“PATH”，“QUERY”，“REF”，“PROTOCOL”，“AUTHORITY”，“FILE”和“USERINFO”。 如果有任一参数为 NULL，则返回 NULL。 例如 parse_url(' http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') 返回 'facebook.com'。 还可以通过提供关键词 string3 作为第三个参数来提取 QUERY 中特定键的值。 例如 parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') 返回 'v1'。
REGEXP(string1, string2)	如果 string1 的任何（可能为空）子字符串与 Java 正则表达式 string2 匹配，则返回 TRUE，否则返回 FALSE。 如果有任一参数为 NULL，则返回 NULL。
REVERSE(string)	返回反转的字符串。如果字符串为 NULL，则返回 NULL。
SPLIT_INDEX(string1, string2, integer1)	通过分隔符 string2 拆分 string1，返回拆分字符串的第 integer（从零开始）个字符串。如果整数为负，则返回 NULL。 如果有任一参数为 NULL，则返回 NULL。
STR_TO_MAP(string1[, string2, string3])	使用分隔符将 string1 拆分为键值对后返回一个 map。string2 是 pair 分隔符，默认为 ‘,’。string3 是键值分隔符，默认为 ‘=’。 pair 分隔符与键值分隔符均为正则表达式，当使用特殊字符作为分隔符时请提前进行转义，例如 <([{\^-=$!|]})?*+.>。
SUBSTR(string[, integer1[, integer2]])	返回字符串的子字符串，从位置 integer1 开始，长度为 integer2（默认到末尾）。

版权所有 © 华为技术有限公司

SQL函数	描述
JSON_VAL(STRING json_string, STRING json_path)	从json形式的字符串json_string中提取指定json_path的值。具体函数使用可以参考**JSON_VAL函数使用说明**说明。 **说明** 以下规则优先级按照顺序从高到低。 1. 不允许json_string和json_path为NULL 2. json_string格式必须为合法的json串，否则函数返回NULL 3. json_string为空字符串，则函数返回空字符串 4. json_path为空字串或路径不存在，则函数返回NULL

JSON_VAL 函数使用说明

- 语法

```
STRING JSON_VAL(STRING json_string, STRING json_path)
```

表 2-83 参数说明

参数	数据类型	说明
json_string	STRING	需要解析的JSON对象，使用字符串表示。
json_path	STRING	解析JSON的路径表达式，使用字符串表示。 目前path支持如下表达式参考下表**表2-84**。

表 2-84 json_path 参数支持的表达式

表达式	说明
$	根对象
[]	数组下标
*	数组通配符
.	取子元素

- 示例

  a. 测试输入数据。

  测试数据源kafka，具体消息内容参考如下：

  ```
  {"name":"James","age":24,"gender":"male","grade":{"math":95,"science":[80,85],"english":100}}
  ```

  b. 使用JSON_VAL编写SQL

  ```
  CREATE TABLE kafkaSource (
    message string
  ) WITH (
    'connector' = 'kafka',
    'topic-pattern' = '<yourSinkTopic>',
    'properties.bootstrap.servers' =
  ```


版权所有 © 华为技术有限公司

```
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv',
  'csv.field-delimiter' = '\u0001',
  'csv.quote-character' = '''
);


CREATE TABLE printSink (
  message1 STRING,
  message2 STRING,
  message3 STRING,
  message4 STRING,
  message5 STRING,
  message6 STRING
) WITH (
  'connector' = 'print'
);
insert into printSink select
JSON_VAL(message,''),
JSON_VAL(message,'$.name'),
JSON_VAL(message,'$.grade.science'),
JSON_VAL(message,'$.grade.science[*]'),
JSON_VAL(message,'$.grade.science[1]'),
JSON_VAL(message,'$.grade.dddd')
from kafkaSource;
```

c. 查看taskmanager的out文件的输出结果

```
+I[null, James, [80,85], [80,85], 85, null]
```

2.7.4.5 时间函数

Flink OpenSource SQL所支持的时间函数如表2-85所示。

函数说明

表 2-85 时间函数

函数	返回值	描述
DATE string	DATE	以“**yyyy-MM-dd**”的形式返回从字符串解析的 SQL 日期。
DATE_ADD	STRING	指定日期增加目标天数后的日期，数据类型为**STRING**。
DATE_SUB	STRING	指定日期减去目标天数后的日期，数据类型为**STRING**。
TIME string	TIME	以“**HH:mm:ss**”的形式返回从字符串解析的 SQL 时间。
TIMESTAMP string	TIMESTAMP	以“**yyyy-MM-dd HH:mm:ss[.SSS]**”的形式返回从字符串解析的 SQL 时间戳。

版权所有 © 华为技术有限公司

函数	返回值	描述
INTERVAL string range	INTERVAL	从“dd hh:mm:ss.fff”形式的字符串解析 SQL 毫秒间隔或者从“yyyy-mm”形式的字符串解析 SQL 月数间隔。 间隔范围可以 是 DAY，MINUTE，DAY TO HOUR 或 DAY TO SECOND，以毫秒为间隔；YEAR 或 YEAR TO MONTH 表示几个月的间隔。 例 如 INTERVAL '10 00:00:00.004' DAY TO SECOND，INTERVAL '10' DAY 或 INTERVAL '2-10' YEAR TO MONTH 返回间隔。
CURRENT_DATE	DATE	返回本地时区中的当前 SQL 日期。在流模式下为每条记录进行取值。 但在批处理模式下，它在查询开始时计算一次，并对每一行使用相同的结果。
CURRENT_TIME	TIME	返回本地时区的当前 SQL 时间，这是 LOCAL_TIME 的同义词。
CURRENT_TIMESTAMP	TIMESTAMP	返回本地时区的当前 SQL 时间戳，返回类型为 TIMESTAMP_LTZ(3)。在流模式下为每条记录进行取值。 但在批处理模式下，它在查询开始时计算一次，并对每一行使用相同的结果。
LOCALTIME	TIME	返回本地时区的当前 SQL 时间，返回类型为 TIME(0)。在流模式下为每条记录进行取值。但在批处理模式下，它在查询开始时计算一次，并对每一行使用相同的结果。
LOCALTIMESTAMP	TIMESTAMP	返回本地时区的当前 SQL 时间，返回类型为 TIMESTAMP(3)。在流模式下为每条记录进行取值。 但在批处理模式下，它在查询开始时计算一次，并对每一行使用相同的结果。
NOW()	TIMESTAMP	返回本地时区的当前 SQL 时间戳，这是 CURRENT_TIMESTAMP 的同义词。
CURRENT_ROW_TIMESTAMP()	TIMESTAMP_LTZ(3)	返回本地时区的当前 SQL 时间戳，返回类型为 TIMESTAMP_LTZ(3)。无论是在批处理模式还是流模式下，都会为每条记录进行取值。
EXTRACT(timeintervalunit FROM temporal)	BIGINT	返回从时间的时间间隔单位部分提取的 long 值。 例如 EXTRACT(DAY FROM DATE '2006-06-05') 返回 5。
YEAR(date)	BIGINT	从 SQL 日期 date 返回年份。相当于 EXTRACT(YEAR FROM date)。例如 YEAR(DATE '1994-09-27') 返回 1994。

版权所有 © 华为技术有限公司

函数	返回值	描述
QUARTER(date)	BIGINT	从 SQL 日期 date 返回一年中的季度（1 到 4 之间的整数）。相当于 EXTRACT(QUARTER FROM date)。 例如 QUARTER(DATE '1994-09-27') 返回 3。
MONTH(date)	BIGINT	从 SQL 日期 date 返回一年中的月份（1 到 12 之间的整数）。相当于 EXTRACT(MONTH FROM date)。 例如 MONTH(DATE '1994-09-27') 返回 9。
WEEK(date)	BIGINT	从 SQL 日期 date 返回一年中的第几周（1 到 53 之间的整数）。相当于 EXTRACT(WEEK FROM date)。 例如 WEEK(DATE '1994-09-27') 返回 39。
DAYOFYEAR(date)	BIGINT	从 SQL 日期 date 返回一年中的第几天（1 到 366 之间的整数）。相当于 EXTRACT(DOY FROM date)。 例如 DAYOFYEAR(DATE '1994-09-27') 返回 270。
DAYOFMONTH(date)	BIGINT	从 SQL 日期 date 返回一个月中的第几天（1 到 31 之间的整数）。相当于 EXTRACT(DAY FROM date)。 例如 DAYOFWEEK(DATE '1994-09-27') 返回 3。
DAYOFWEEK(date)	BIGINT	计算当前日期是当前周的第几天 其中周日设为1 例如：DAYOFWEEK(DATE '1994-09-27') 返回3
HOUR(timestamp)	BIGINT	从 SQL 时间戳 timestamp 返回小时单位部分的小时（0 到 23 之间的整数）数。相当于 EXTRACT(HOUR FROM timestamp)。 例如 MINUTE(TIMESTAMP '1994-09-27 13:14:15') 返回 14。
MINUTE(timestamp)	BIGINT	从 SQL 时间戳 timestamp 返回分钟单位的分钟数（0 到 59 之间的整数）。相当于 EXTRACT(MINUTE FROM timestamp)。 例如 MINUTE(TIMESTAMP '1994-09-27 13:14:15') 返回 14。
SECOND(timestamp)	BIGINT	从 SQL 时间戳 timestamp 返回秒单位部分的秒数（0 到 59 之间的整数）。相当于 EXTRACT(SECOND FROM timestamp)。 例如 SECOND(TIMESTAMP '1994-09-27 13:14:15') 返回 15。

版权所有 © 华为技术有限公司

函数	返回值	描述
FLOOR(timepoint TO timeintervalunit)	TIME	返回将时间点 timepoint 向下取值到时间单位 timeintervalunit 的值。例如 FLOOR(TIME '12:44:31' TO MINUTE) 返回 12:44:00。
CEIL(timepoint TO timeintervalunit)	TIME	返回将时间点 timepoint 向上取值到时间单位 TIMEINTERVALUNIT 的值。 例如 CEIL(TIME '12:44:31' TO MINUTE) 返回 12:45:00。
(timepoint1, temporal1) OVERLAPS (timepoint2, temporal2)	BOOLEAN	如果由 (timepoint1, temporal1) 和 (timepoint2, temporal2) 定义的两个时间间隔重叠，则返回 TRUE。 时间值可以是时间点或时间间隔。例如 (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) 返回 TRUE； (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) 返回 FALSE。
DATE_FORMAT(timestamp, string)	STRING	将时间戳 timestamp 转换为日期格式字符串 string 指定格式的字符串值。格式字符串与 Java 的 SimpleDateFormat 兼容。
TIMESTAMPADD(timeintervalunit, interval, timepoint)	TIMESTAMP/ DATE/ TIME	将整型interval与timeintervalunit组成的结果添加日期或日期时间到timepoint中，并返回添加后的日期时间 例如：TIMESTAMPADD(WEEK, 1, DATE '2003-01-02') 返回2003-01-09
TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)	INT	返回 timepoint1 和 timepoint2 之间时间间隔。间隔的单位由第一个参数给出，它应该是以下值之一： SECOND，MINUTE，HOUR，DAY，MONTH 或 YEAR。
CONVERT_TZ(string1, string2, string3)	TIMESTAMP	日期时间 string1（具有默认 ISO 时间戳格式‘yyyy-MM-dd HH:mm:ss’）从时区 string2 转换为时区 string3 的值。 时区的格式应该是缩写如“PST”，全名如“Country A/City A”，或自定义 ID 如“GMT-08:00”。 例如 CONVERT_TZ('1970-01-01 00:00:00', 'UTC', 'Country A/City A') 返回 '1969-12-31 16:00:00’ 。

版权所有 © 华为技术有限公司

函数	返回值	描述
FROM_UNIXTIME(numeric[, string])	STRING	以字符串格式 string 返回数字参数 numberic 的表示形式（默认为 ‘yyyy-MM-dd HH:mm:ss’ ）。numeric 是一个内部 时间戳值，表示自'1970-01-01 00:00:00’ UTC 以来的秒数，由 UNIX_TIMESTAMP() 函数生成。返回值以会话时区表示 （ 在 TableConfig 中指定 ）。 例如，如果在 UTC 时区，FROM_UNIXTIME(44) 返回 ‘1970-01-01 00:00:44’，如果在 ‘Asia/Tokyo’ 时区，则返回 ‘1970-01-01 09:00:44’ 。
UNIX_TIMESTAMP()	BIGINT	以秒为单位获取当前的 Unix 时间戳。此函数不是确定性的，这意味着将为每个记录重新计算该值。
UNIX_TIMESTAMP(string1[, string2])	BIGINT	使用表配置中指定的时区将格式为 string2 的日期时间字符串 string1（ 如果未指定默认情况下：yyyy-MM-dd HH:mm:ss ） 转换为 Unix 时间戳（ 以秒为单位 ）。
TO_DATE(string1[, string2])	DATE	将格式为 string2（ 默认为 ‘yyyy-MM-dd’ ）的字符串 string1 转换为日期。
TO_TIMESTAMP_LTZ(numeric, precision)	TIMESTAMP_LTZ	将纪元秒或纪元毫秒转换为 TIMESTAMP_LTZ，有效精度为 0 或 3，0 代表 TO_TIMESTAMP_LTZ(epochSeconds, 0)，3 代表 TO_TIMESTAMP_LTZ(epochMilliseconds, 3)。
TO_TIMESTAMP(string1[, string2])	TIMESTAMP	将 ‘UTC+0’ 时区下格式为 string2（ 默认为：‘yyyy-MM-dd HH:mm:ss’ ）的字符串 string1 转换为时间戳。
CURRENT_WATERMARK(rowtime)	-	返回给定时间列属性rowtime 的当前水印，如果管道中的当前操作没有可用的上游操作的公共水印时则为 NULL。 函数的返回类型被推断为与提供的时间列属性匹配，但调整后的精度为 3。例如时间列属性为 TIMESTAMP_LTZ(9)，则函数将返回 TIMESTAMP_LTZ(3)。 请注意，此函数可以返回 NULL，您可能必须考虑这种情况。例如，如果您想过滤掉后期数据，您可以使用： `WHERE CURRENT_WATERMARK(ts) IS NULL OR ts > CURRENT_WATERMARK(ts)`

版权所有 © 华为技术有限公司

DATE

- **功能描述**

  DATE函数将**"yyyy-MM-dd"**日期格式的字符串解析为DATE类型的日期。

- **语法说明**

```
DATE DATE string
```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	SQL日期格式的字符串。 注意该字符串的格式必须为"yyyy-MM-dd"格式，否则语义校验会报错。

- **示例**

  – 测试语句

```
SELECT
   DATE "2021-08-19" AS `result`
FROM
   testtable;
```

  – 测试结果

result
2021-08-19

DATE_ADD

- **功能描述**

  DATE_ADD函数返回指定日期增加目标天数后的日期。

- **语法说明**

```
DATE_ADD(string startdate, int days)
```

- **入参说明**

  – startdate 指定时间，数据类型为TIMESTAMP或者STRING。

  > **说明**
  >
  > STRING类型日期格式为yyyy-MM-dd HH:mm:ss。
  >
  > 支持参数为NULL的特殊情况处理

  – days 目标天数，数据类型为INT。

- **返回值**

  指定日期增加目标天数后的日期，数据类型为STRING。

- **示例**

  提交FlinkSQL语句

```
CREATE TABLE source (
  time1 TIMESTAMP
) WITH (
  'connector' = 'datagen',
```

 版权所有 © 华为技术有限公司

```
  'rows-per-second' = '1'
);
create table Sink (
  date1 string,
  date2 string,
  date3 string
) with ('connector' = 'print');
INSERT into
  Sink
select
  DATE_ADD(time1, 30) as date1,
  DATE_ADD('2017-09-15 00:00:00', 30) as date2,
  DATE_ADD(cast(null as timestamp),30) as date3
FROM source
```

测试结果

date1 (string)	date2 (string)	date3 (string)
2024-06-28	2017-10-15	null

DATE_SUB

- **功能描述**

  DATE_SUB函数返回指定日期减去目标天数后的日期。

- **语法说明**

  ```
  DATE_SUB(string startdate, int days)
  ```

- **入参说明**
  - startdate 指定时间，数据类型为TIMESTAMP或者STRING。

    说明

    STRING类型日期格式为yyyy-MM-dd HH:mm:ss。

    支持参数为NULL的特殊情况处理
  - days 目标天数，数据类型为INT。
- **返回值**

  指定日期减去目标天数后的日期，数据类型为STRING。

- **示例**

  提交FlinkSQL语句

  ```
  CREATE TABLE source (
    time1 TIMESTAMP
  ) WITH (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );
  create table Sink (
    date1 string,
    date2 string,
    date3 string
  ) with ('connector' = 'print');
  INSERT into
    Sink
  select
    DATE_SUB(time1,30) as date1,
    DATE_SUB('2017-09-15 00:00:00', 30) as date2,
    DATE_SUB(cast(null as timestamp),30) as date3
  FROM source
  ```

  测试结果

date1 (string)	date2 (string)	date3 (string)
2024-04-29	2017-08-16	null

TIME

- **功能描述**

  将时间字符串以**"HH:mm:ss[.fff]"**形式解析为SQL时间，结果以TIME类型返回。

- **语法说明**

```
TIME TIME string
```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间字符串。 注意该字符串格式必须**"HH:mm:ss[.fff]"**，否则语义校验会报错。

- **示例**
  - 测试语句

```
SELECT
    TIME "10:11:12" AS `result`,
        TIME "10:11:12.032" AS `result2`
FROM
    testtable;
```

  - 测试结果

result	result2
10:11:12	10:11:12.032

TIMESTAMP

- **功能描述**

  将时间字符串转换为时间戳，时间字符串格式为："**yyyy-MM-dd HH:mm:ss[.fff]**"，以TIMESTAMP(3)类型返回。

- **语法说明**

```
TIMESTAMP(3) TIMESTAMP string
```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间戳字符串。 注意该字符串格式必须为"**yyyy-MM-dd HH:mm:ss[.fff]**"，否则语义校验会报错。

版权所有 © 华为技术有限公司

- **示例**
  - 测试语句

```
SELECT
    TIMESTAMP "1997-04-25 13:14:15" AS `result`,
        TIMESTAMP "1997-04-25 13:14:15.032" AS `result2`
FROM
    testtable;
```

  - 测试结果

result	result2
1997-04-25 13:14:15	1997-04-25 13:14:15.032

INTERVAL

- **功能描述**

  INTERVAL函数用于表示时间间隔。

- **语法说明**

```
INTERVAL INTERVAL string range
```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间戳字符串，搭配参数range使用。两种格式类型，分别为： ● 一种为**"yyyy-MM"**即保存年份和月份，精度到月份，它的range参数可以为**YEAR**或者**YEAR To Month**。 ● 一种为天时间**"dd HH:mm:sss.fff"**，用来保存天数、小时、分钟、秒和毫秒， 精度最低到毫秒。它的range参数可以为**DAY**、**MINUTE**、**DAY TO HOUR**、**DAY TO SECOND**。
range	INTERVAL	时间间隔说明，搭配string参数使用，详细请参考string参数说明。 取值范围为：**YEAR**、**YEAR To Month**、**DAY**、**MINUTE**、**DAY TO HOUR**、**DAY TO SECOND**。

- **示例**

  测试语句

```
--表示间隔10天4毫秒。
INTERVAL '10 00:00:00.004' DAY TO second
--DAY表示间隔10天
INTERVAL '10'
--表示间隔2年10个月
INTERVAL '2-10' YEAR TO MONTH
```

CURRENT_DATE

- **功能描述**

 版权所有 © 华为技术有限公司

以UTC时区**"yyyy-MM-dd"**格式返回当前SQL日期，返回类型为DATE。

- **语法说明**
  ```
  DATE CURRENT_DATE
  ```
- **入参说明**

  无。
- **示例**
  - 测试语句
    ```
    SELECT
      CURRENT_DATE AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
2021-10-28

CURRENT_TIME

- **功能描述**

  以UTC（UTC+0）时区“HH:mm:sss.fff”格式返回当前SQL时间，返回类型为TIME。
- **语法说明**
  ```
  TIME CURRENT_TIME
  ```
- **入参说明**

  无。
- **示例**
  - 测试语句
    ```
    SELECT
      CURRENT_TIME AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
08:29:19.289

CURRENT_TIMESTAMP

- **功能描述**

  以UTC（UTC+0）时区返回当前SQL时间戳，返回类型为TIMESTAMP(3)。
- **语法说明**
  ```
  TIMESTAMP(3) CURRENT_TIMESTAMP
  ```
- **入参说明**

  无。
- **示例**

 版权所有 © 华为技术有限公司

- 测试语句

```
SELECT
  CURRENT_TIMESTAMP AS `result`
FROM
  testtable;
```

- 测试结果

result
2021-10-28 08:33:51.606

LOCALTIME

- **功能描述**

  返回当前时区的当前SQL时间，返回类型为TIME。

- **语法说明**

```
TIME LOCALTIME
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  LOCALTIME AS `result`
FROM
  testtable;
```

  - 测试结果

result
16:39:37.706

LOCALTIMESTAMP

- **功能描述**

  返回当前时区的当前SQL时间戳，返回类型为TIMESTAMP(3)。

- **语法说明**

```
TIMESTAMP(3) LOCALTIMESTAMP
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  LOCALTIMESTAMP AS `result`
FROM
  testtable;
```

  - 测试结果

result
2021-10-28 16:43:17.625

 版权所有 © 华为技术有限公司

EXTRACT

- **功能描述**

  提取时间点或时间间隔中指定某一时间单位的部分，以BIGINT类型返回。

- **语法说明**

  ```
  BIGINT EXTRACT(timeinteravlunit FROM temporal)
  ```

- **入参说明**

参数名	数据类型	参数说明
timeinteravlunit	TIMEUNIT	需要从时间点或时间间隔中提取的时间单位，取值可以是：YEAR/QUARTER/MONTH/WEEK/DAY/DOY/HOUR/MINUTE/SECOND。
temporal	DATE/TIME/TIMESTAMP/INTERVAL	时间点或时间间隔。

注意

不允许指定不存在于时间点或时间间隔中的时间单位，否则作业会提交失败。

例如如下错误语句，会报错YEAR不能从TIME中提取。

```
SELECT
   EXTRACT(YEAR FROM TIME '12:44:31' ) AS `result`
FROM
   testtable;
```

- **示例**

  - 测试语句

    ```
    SELECT
       EXTRACT(YEAR FROM DATE '1997-04-25' ) AS `result`,
          EXTRACT(MINUTE FROM TIME '12:44:31') AS `result2`,
          EXTRACT(SECOND FROM TIMESTAMP '1997-04-25 13:14:15') AS `result3`,
          EXTRACT(YEAR FROM INTERVAL '2-10' YEAR TO MONTH) AS `result4`,
    FROM
       testtable;
    ```

  - 测试结果

result	result2	result3	result4
1997	44	15	2

YEAR

- **功能描述**

  从SQL日期date返回年份，以BIGINT类型返回。

- **语法说明**

  ```
  BIGINT YEAR(date)
  ```


版权所有 © 华为技术有限公司

- **入参说明**

参数名	数据类型	参数说明
date	DATE	DATE类型的SQL日期。

- **示例**
  - 测试语句

```
SELECT
    YEAR(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

  - 测试结果

result
1997

QUARTER

- **功能描述**

  从SQL日期返回表示该日期季度的数字（1到4之间的整数），返回类型为BIGINT。

- **语法说明**

```
BIGINT QUARTER(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句

```
SELECT
    QUARTER(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

  - 测试结果

result
2

MONTH

- **功能描述**

  返回输入时间的月份（1到12之间的整数），返回类型为BIGINT。

- **语法说明**

```
BIGINT MONTH(date)
```

 版权所有 © 华为技术有限公司

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句

```
SELECT
    MONTH(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

  - 测试结果

result
4

WEEK

- **功能描述**

  计算当前日期是一年中的第几周，以BIGINT类型返回。

- **语法说明**

```
BIGINT WEEK(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句

```
SELECT
    WEEK(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

  - 测试结果

result
17

DAYOFYEAR

- **功能描述**

  计算当前日期是一年中的第几天（返回1到366 之间的整数），以BIGINT类型返回。

- **语法说明**

```
BIGINT DAYOFYEAR(date)
```

 版权所有 © 华为技术有限公司

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句

```
SELECT
    DAYOFYEAR(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

  - 测试结果

result
115

DAYOFMONTH

- **功能描述**

  计算当前日期是这个月的第几天（1到31之间的整数），以BIGINT类型返回。

- **语法说明**

```
BIGINT DAYOFMONTH(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句

```
SELECT
    DAYOFMONTH(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

  - 测试结果

result
25

DAYOFWEEK

- **功能描述**

  计算当前日期是当前周的第几天（1 到 7之间的整数），以BIGINT类型返回。

  说明

  需要注意这里自然周的起点是星期天，即每周的第1天是星期天，第2天是星期一，以此类推。

 版权所有 © 华为技术有限公司

- **语法说明**
  BIGINT **DAYOFWEEK**(date)
- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句
    ```
    SELECT
      DAYOFWEEK(DATE '1997-04-25') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
6

HOUR

- **功能描述**
  从当前时间戳获取以24小时制的小时数进行返回，范围0-23（0 到 23 之间的整数），返回类型为BIGINT。
- **语法说明**
  BIGINT **HOUR**(timestamp)
- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
      HOUR(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
10

MINUTE

- **功能描述**
  返回当前时间戳中的分钟数（0 到 59 之间的整数），返回类型为BIGINT。

 版权所有 © 华为技术有限公司

- **语法说明**
  BIGINT **MINUTE**(timestamp)
- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
      MINUTE(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
11

SECOND

- **功能描述**
  返回当前时间戳中的秒数（0 到 59 之间的整数），返回类型为BIGINT。
- **语法说明**
  BIGINT **SECOND**(timestamp)
- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
      SECOND(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
12

FLOOR

- **功能描述**
  返回将时间点向下取值到指定时间单位的值。
- **语法说明**
  TIME/TIMESTAMP(3) **FLOOR**(timepoint **TO** timeintervalunit)

 版权所有 © 华为技术有限公司

- **入参说明**

参数名	数据类型	参数说明
timepoint	TIMESTAMP /TIME	SQL时间或SQL时间戳。
timeintervalunit	TIMEUNIT	时间单位，类型可以是YEAR/QUARTER/ MONTH/WEEK/DAY/DOY/HOUR/MINUTE/ SECOND。

- **示例**
  - 测试语句。

```
SELECT
   FLOOR(TIME '13:14:15' TO MINUTE) AS `result`
      FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
      FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`
FROM    testtable;
```

  - 测试结果

message	message2	message3
13:14	13:14	1997-04-25T13:14

CEIL

- **功能描述**

  返回将时间点向上取值到指定时间单位的值。

- **语法说明**

```
TIME/TIMESTAMP(3) CEIL(timepoint TO timeintervalunit)
```

- **入参说明**

参数名	数据类型	参数说明
timepoint	TIMESTAMP /TIME	SQL时间或SQL时间戳。
timeintervalunit	TIMEUNIT	时间单位，类型可以是YEAR/QUARTER/ MONTH/WEEK/DAY/DOY/HOUR/MINUTE/ SECOND。

- **示例**
  - 测试语句。

```
SELECT
   CEIL(TIME '13:14:15' TO MINUTE) AS `result`
      CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
      CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`
FROM    testtable;
```

  - 测试结果

 版权所有 © 华为技术有限公司

result	result2	result3
13:15	13:15	1997-04-25T13:15

OVERLAPS

- **功能描述**

  如果两个时间范围有重叠，则返回TRUE，反之，则返回FALSE。

- **语法说明**

  ```
  BOOLEAN (timepoint1, temporal1) OVERLAPS (timepoint2, temporal2)
  ```

- **入参说明**

参数名	数据类型	参数说明
timepoint1/ timepoint2	DATE/TIME/ TIMESTAMP	时间点。
temporal1/ temporal2	DATE/TIME/ TIMESTAMP/ INTERVAL	时间点或时间间隔。

  说明

  - (timepoint, temporal)在判断是否重叠时为闭区间。
  - temporal可以是DATE/TIME/TIMESTAMP也可以是INTERVAL。
    - 当temporal是DATE/TIME/TIMESTAMP时，(timepoint, temporal)表示timepoint, temporal之间的时间间隔。允许temporal在timepoint之前，如(DATE '1997-04-25', DATE '1997-04-23')也合法。
    - 当temporal是INTERVAL时，(timepoint, temporal)表示timepoint, timepoint +temporal之间的时间间隔。
  - 必须保证(timepoint1, temporal1)和(timepoint2, temporal2)是同一数据类型的时间间隔。

- **示例**

  - 测试语句

    ```
    SELECT
       (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS
    `result`,
          (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS
    `result2`,
       (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:31:00', INTERVAL '2' HOUR) AS
    `result3`,
       (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:00:00', INTERVAL '3' HOUR) AS
    `result4`,
       (TIMESTAMP '1997-04-25 12:00:00', TIMESTAMP '1997-04-25 12:20:00') OVERLAPS
    (TIMESTAMP '1997-04-25 13:00:00', INTERVAL '2' HOUR) AS `result5`,
       (DATE '1997-04-23', INTERVAL '2' DAY) OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
    AS `result6`,
       (DATE '1997-04-25', DATE '1997-04-23') OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
    AS `result7`
    FROM
       testtable;
    ```

  - 测试结果

版权所有 © 华为技术有限公司

result	result2	result3	result4	result5	result6	result7
true	true	false	true	false	true	true

DATE_FORMAT

- **功能描述**

  将时间戳或时间戳格式的字符串转换为指定格式的日期字符串。

- **语法说明**

  ```
  STRING DATE_FORMAT(timestamp, dateformat)
  ```

- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP/ STRING	时间点。
dateformat	STRING	日期格式字符串。

- **示例**

  - 测试语句

    ```
    SELECT
        DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd HH:mm:ss') AS `result`,
            DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result2`,
        DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yy/MM/dd HH:mm') AS `result3`,
            DATE_FORMAT('1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result4`
    FROM    testtable;
    ```

  - 测试结果

result	result2	result3	result4
1997-04-25 10:11:12	1997-04-25	97/04/25 10:11	1997-04-25

TIMESTAMPADD

- **功能描述**

  参考语法说明，本函数功能为将整型interval与timeintervalunit组成的结果添加到timepoint中，并返回添加后的日期时间。

  说明

  TIMESTAMPADD函数返回结果与timepoint相同。例外场景为：如果timepoint输入类型为TIMESTAMP，也可以将TIMESTAMPADD函数返回结果插入到DATE类型的表字段中。

- **语法说明**

  ```
  TIMESTAMP(3)/DATE/TIME TIMESTAMPADD(timeintervalunit, interval, timepoint)
  ```

- **入参说明**

 版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
timeintervalunit	TIMEUNIT	时间单位。
interval	INT	整型的时间间隔。
timepoint	TIMESTAMP/DATE/TIME	时间点

- **示例**
  - 测试语句

```
SELECT
   TIMESTAMPADD(WEEK, 1, DATE '1997-04-25') AS `result`,
      TIMESTAMPADD(QUARTER, 1, TIMESTAMP '1997-04-25 10:11:12') AS `result2`,
   TIMESTAMPADD(SECOND, 2, TIME '10:11:12') AS `result3`
FROM    testtable;
```

  - 测试结果

result	result2	result3
1997-05-02	• 如果该字段插入到TIMESTAMP类型的表字段中，则返回：1997-07-25T10:11:12 • 如果该字段插入到TIMESTAMP类型的表字段中，则返回：1997-07-25	10:11:14

TIMESTAMPDIFF

- **功能描述**

  参考语法说明，本函数功能为返回timepoint1和timepoint2之间的时间间隔，间隔的单位由第一个参数timepointunit指定。

- **语法说明**

```
INT TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)
```

- **入参说明**

参数名	数据类型	参数说明
timepointunit	TIMEUNIT	时间单位。取值范围为： SECOND、MINUTE、HOUR、DAY、MONTH、YEAR。
timepoint1/timepoint2	TIMESTAMP/DATE	时间点。

- **示例**

 版权所有 © 华为技术有限公司

- 测试语句

```
SELECT
   TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-25 10:00:00', TIMESTAMP '1997-04-28 10:00:00')
AS `result`,
      TIMESTAMPDIFF(DAY, DATE '1997-04-25', DATE '1997-04-28') AS `result2`,
   TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-27 10:00:20', TIMESTAMP '1997-04-25 10:00:00')
AS `result3`
FROM    testtable;
```

- 测试结果

result	result2	result3
3	3	-2

CONVERT_TZ

- **功能描述**

  参考语法说明，本函数将日期时间string1（具有默认ISO时间戳格式'yyyy-MM-dd HH:mm:ss'）从时区string2转换为时区string3的值，结果以STRING类型返回。

- **语法说明**

```
STRING CONVERT_TZ(string1, string2, string3)
```

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串，不符合格式的字符串会返回NULL。
string2	STRING	转换前时区。时区的格式应该是缩写如“PST”，全名如“Country A/City A”，或自定义ID如“GMT-08:00”。
string3	STRING	转换后时区。时区的格式应该是缩写如“PST”，全名如“Country A/City A”，或自定义ID如“GMT-08:00”。

- **示例**

  - 测试语句

```
SELECT
   CONVERT_TZ(1970-01-01 00:00:00, UTC, Country A/City A) AS `result`,
      CONVERT_TZ(1997-04-25 10:00:00, UTC, GMT-08:00) AS `result2`
FROM    testtable;
```

  - 测试结果

result	result2
1969-12-31 16:00:00	1997-04-25 02:00:00

FROM_UNIXTIME

- **功能描述**

  参考语法说明，本函数根据时间戳numeric和当前时区返回string格式的时间。

 版权所有 © 华为技术有限公司

- **语法说明**
  STRING **FROM_UNIXTIME**(numeric[, string])
- **入参说明**

参数名	数据类型	参数说明
numeric	BIGINT	内部时间戳值，表示自'1970-01-01 00:00:00' UTC 以来的秒数，值可以由 UNIX_TIMESTAMP() 函数生成。
string	STRING	时间字符串格式。如果该参数不指定，则默认为'yyyy-MM-dd HH:mm:ss'。

- **示例**
  - 测试语句

```
SELECT
    FROM_UNIXTIME(44) AS `result`,
        FROM_UNIXTIME(44, 'yyyy:MM:dd') AS `result2`
FROM    testtable;
```

  - 测试结果

result	result2
1970-01-01 08:00:44	1970:01:01

UNIX_TIMESTAMP

- **功能描述**
  以秒为单位获取当前的Unix时间戳。以BIGINT类型返回。
- **语法说明**
  BIGINT **UNIX_TIMESTAMP**()
- **入参说明**
  无。
- **示例**
  - 测试语句

```
SELECT
    UNIX_TIMESTAMP() AS `result`
FROM
    table;
```

  - 测试结果

result
1635401982

UNIX_TIMESTAMP(string1[, string2])

- **功能描述**
  参数语法说明，本函数将以string2格式的时间字符串string1转为Unix 时间戳（以秒为单位）。以BIGINT类型返回。

 版权所有 © 华为技术有限公司

- **语法说明**
  BIGINT **UNIX_TIMESTAMP**(string1[, string2])
- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合string2参数格式的字符串语法会报错。
string2	STRING	时间字符串格式。如果不指定该参数，则默认为'yyyy-MM-dd HH:mm:ss' 。

- **示例**
  - 测试语句

```
SELECT
    UNIX_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
        UNIX_TIMESTAMP('1997-04-25 00:00:10', 'yyyy-MM-dd HH:mm:ss') AS `result2`,
        UNIX_TIMESTAMP('1997-04-25 00:00:00') AS `result3`
FROM
    testtable;
```

  - 测试结果

result	result2	result3
861897600	861897610	861897600

TO_DATE

- **功能描述**
  参数语法说明，本函数将string2格式的日期字符串string1转换为DATE类型。
- **语法说明**
  DATE **TO_DATE**(string1[, string2])
- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合格式的字符串会执行报错。
string2	STRING	字符串格式。如果不指定该参数，则默认为'yyyy-MM-dd' 。

- **示例**
  - 测试语句

```
SELECT
    TO_DATE('1997-04-25') AS `result`,
        TO_DATE('1997:04:25', 'yyyy-MM-dd') AS `result2`,
        TO_DATE('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
    testtable;
```

  - 测试结果

 版权所有 © 华为技术有限公司

result	result2	result3
1997-04-25	1997-04-25	1997-04-25

TO_TIMESTAMP

- **功能描述**

  将string2格式的日期时间字符串string1转换为TIMESTAMP类型返回。

- **语法说明**

```
TIMESTAMP TO_TIMESTAMP(string1[, string2])
```

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合格式的字符串会返回NULL。
string2	STRING	日期字符串格式。如果该参数不指定，则默认为'yyyy-MM-dd HH:mm:ss' 。

- **示例**
  - 测试语句

```
SELECT
    TO_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
        TO_TIMESTAMP('1997-04-25 00:00:00') AS `result2`,
        TO_TIMESTAMP('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
    testtable;
```

  - 测试结果

result	result2	result3
1997-04-25 00:00	1997-04-25 00:00	1997-04-25 00:00

 版权所有 © 华为技术有限公司

2.7.4.6 条件函数

函数说明

表 2-86 条件函数

条件函数	函数说明
CASE value WHEN value1_1 [, value1_2]* THEN result1 [WHEN value2_1 [, value2_2]* THEN result2]* [ELSE resultZ] END	当第一个时间值包含在 (valueX_1, valueX_2, …) 中时，返回 resultX。当没有值匹配时，如果提供则返回 result_z， 否则返回 NULL。
CASE WHEN condition1 THEN result1 [WHEN condition2 THEN result2]* [ELSE resultZ] END	满足第一个条件 X 时返回 resultX。当不满足任何条件时，如果提供则返回 result_z，否则返回 NULL。
NULLIF(value1, value2)	如果 value1 等于 value2 返回 NULL；否则返回 value1。 例如 NULLIF(5, 5) 返回 NULL；NULLIF(5, 0) 返回 5。
COALESCE(value1, value2 [, value3]*)	从 value1, value2, … 返回第一个不为 NULL 的值。 例如 COALESCE(3, 5, 3) 返回 3。
IF(condition, true_value, false_value)	如果满足条件，则返回 true_value，否则返回 false_value。 例如 IF(5 > 3, 5, 3) 返回 5。
IFNULL(input, null_replacement)	如果输入为 NULL，则返回 null_replacement；否则返回输入。与 COALESCE 或 CASE WHEN 相比，此函数返回的数据类型 在是否为空方面非常明确。。返回的类型是两个参数的公共类型，但只有在 null_replacement 可为空时才能为空。该函数允许将可 为空的列传递到使用 NOT NULL 约束声明的函数或表中。 例如 IFNULL(nullable_column, 5) 一定不返回 NULL。
IS_ALPHA(string)	如果字符串中的所有字符都是字母，则返回 true，否则返回 false。

版权所有 © 华为技术有限公司

条件函数	函数说明
IS_DECIMAL(string)	如果 string 可以解析为有效数字，则返回 true，否则返回 false。
IS_DIGIT(string)	如果字符串中的所有字符都是数字，则返回 true，否则返回 false。
GREATEST(value1[, value2]*)	返回所有输入参数的最大值，如果输入参数中包含 NULL，则返回 NULL。
LEAST(value1[, value2]*)	返回所有输入参数的最小值，如果输入参数中包含 NULL，则返回 NULL。

2.7.4.7 类型转换函数

表 2-87 类型转换函数

SQL函数	描述
CAST(value AS type)	返回被强制转换为类型 type 的新值。 例如 CAST('42' AS INT) 返回 42； CAST(NULL AS VARCHAR) 返回 VARCHAR 类型的 NULL。
TYPEOF(input) | TYPEOF(input, force_serializable)	返回输入表达式的数据类型的字符串表示形式。默认情况下返回的字符串是一个摘要字符串，可能会为了可读性而省略某些细节。 如果 force_serializable 设置为 TRUE，则字符串表示可以保留在目录中的完整数据类型。请注意， 特别是匿名的内联数据类型没有可序列化的字符串表示。在这种情况下返回 NULL。

CAST 语法格式

```
CAST(value AS type)
```

CAST 语法说明

类型强制转换。

CAST 注意事项

如果输入为NULL，则返回NULL。

CAST 示例一：将 amount 值转换成整型

将amount值转换成整型。

```
insert into temp select cast(amount as INT) from source_stream;
```

 版权所有 © 华为技术有限公司

表 2-88 CAST 类型转换函数示例

示例	说明	示例
cast(v1 as string)	将v1转换为字符串类型，v1可以是数值类型，TIMESTAMP/DATE/TIME。	表T1: | content (INT)

5	 语句： SELECT cast(content as varchar) FROM T1; 结果： "5"
cast (v1 as int)	将v1转换为int, v1可以是数值类型或字符类。	表T1: | content (STRING)

"5"	 语句： SELECT cast(content as int) FROM T1; 结果： 5
cast(v1 as timestamp)	将v1转换为timestamp类型，v1可以是字符串或DATE/TIME。	表T1: | content (STRING)

"2018-01-01 00:00:01"	 语句： SELECT cast(content as timestamp) FROM T1; 结果： 1514736001000
cast(v1 as date)	将v1转换为date类型， v1可以是字符串或者TIMESTAMP。	表T1: | content (TIMESTAMP)

1514736001000	 语句： SELECT cast(content as date) FROM T1; 结果： "2018-01-01"

说明

Flink作业不支持使用CAST将“BIGINT”转换为“TIMESTAMP”，可以使用to_timestamp进行转换。

版权所有 © 华为技术有限公司

CAST 示例二

1. 参考**Kafka**和**Print**创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   CREATE TABLE kafkaSource (
     cast_int_to_string int,
     cast_String_to_int string,
     case_string_to_timestamp string,
     case_timestamp_to_date timestamp
   ) WITH (
     'connector' = 'kafka',
     'topic' = '***KafkaTopic***',
     'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
     'properties.group.id' = '***GroupId***',
     'scan.startup.mode' = 'latest-offset',
     "format" = "json"
   );

   CREATE TABLE printSink (
     cast_int_to_string string,
     cast_String_to_int int,
     case_string_to_timestamp timestamp,
     case_timestamp_to_date date
   ) WITH (
     'connector' = 'print'
   );

   insert into printSink select
     cast(cast_int_to_string as string),
     cast(cast_String_to_int as int),
     cast(case_string_to_timestamp as timestamp),
     cast(case_timestamp_to_date as date)
   from kafkaSource;
   ```

2. 连接Kafka集群，向Kafka的topic中发送如下测试数据：

   ```
   {"cast_int_to_string":"1", "cast_String_to_int": "1", "case_string_to_timestamp": "2022-04-02 15:00:00",
   "case_timestamp_to_date": "2022-04-02 15:00:00"}
   ```

3. 查看输出结果:

   - 方法一：
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
     iii. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
   - 方法二：如果在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
     iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   查询结果参考如下：

   ```
   +I(1,1,2022-04-02T15:00,2022-04-02)
   ```


版权所有 © 华为技术有限公司

2.7.4.8 集合函数

函数说明

表 2-89 集合函数说明

集合函数	函数说明
CARDINALITY(array)	返回数组中元素的数量。
array '[' INT ']'	返回数组中 INT 位置的元素。索引从 1 开始。
ELEMENT(array)	返回数组的唯一元素（其基数应为 1）；如果数组为空，则返回 NULL。如果数组有多个元素，则抛出异常。
CARDINALITY(map)	返回 map 中的 entries 数量。
map ‘[’ value ‘]’	返回 map 中指定 key 对应的值。

2.7.4.9 JSON 函数

JSON函数使用SQL标准的ISO/IEC TR 19075-6中描述的JSON路径表达式。它们的语法受到ECMAScript的启发并采用了ECMAScript的许多特性，但既不是其子集，也不是其超集。

路径表达式有两种，一种是宽松模式，另一种是严格模式。当省略时，它默认为严格模式。严格模式旨在从模式的角度检查数据，当数据不符合路径表达式时将抛出错误。但是，像JSON_VALUE这样的函数允许在遇到错误时定义回退行为。但是宽松模式会将错误转换为空序列。

特殊字符$表示JSON路径中的根节点。路径可以访问属性（$.a）、数组元素（$.a[0].b）或数组中的所有元素（$.a[*].b）。

已知限制：当前并非所有宽松模式的特性都得到了正确的支持。

版权所有 © 华为技术有限公司

表 2-90 JSON 函数

SQL函数	描述
IS JSON [{ VALUE | SCALAR | ARRAY | OBJECT }]	判断给定的字符串是否是有效的JSON字符串。 指定可选类型参数会对允许类型的JSON对象施加约束。如果字符串是有效的JSON，但不是该类型，则返回false。默认值为VALUE。 -- TRUE '1' IS JSON '[]' IS JSON '{}' IS JSON -- TRUE '"abc"' IS JSON -- FALSE 'abc' IS JSON NULL IS JSON -- TRUE '1' IS JSON SCALAR -- FALSE '1' IS JSON ARRAY -- FALSE '1' IS JSON OBJECT -- FALSE '{}' IS JSON SCALAR -- FALSE '{}' IS JSON ARRAY -- TRUE '{}' IS JSON OBJECT
JSON_EXISTS(jsonValue, path [{ TRUE | FALSE | UNKNOWN | ERROR } ON ERROR])	判断JSON字符串是否满足给定的路径搜索条件。 如果忽略错误行为，则FALSE ON ERROR为默认值。 -- TRUE SELECT JSON_EXISTS('{"a": true}', '$.a'); -- FALSE SELECT JSON_EXISTS('{"a": true}', '$.b'); -- TRUE SELECT JSON_EXISTS('{"a": [{ "b": 1 }]}', '$.a[0].b'); -- TRUE SELECT JSON_EXISTS('{"a": true}', 'strict $.b' TRUE ON ERROR); -- FALSE SELECT JSON_EXISTS('{"a": true}', 'strict $.b' FALSE ON ERROR);
JSON_STRING(value)	将该值序列化为JSON。 此函数返回包含序列化值的JSON字符串。如果值为NULL，则函数返回NULL。 -- NULL JSON_STRING(CAST(NULL AS INT)) -- '1' JSON_STRING(1) -- 'true' JSON_STRING(TRUE) -- '"Hello, World!"' JSON_STRING('Hello, World!') -- '[1,2]' JSON_STRING(ARRAY[1, 2])

版权所有 © 华为技术有限公司

SQL函数	描述

JSON_VALUE(jsonValue, path [RETURNING <dataType>] [{ NULL | ERROR | DEFAULT <defaultExpr> } ON EMPTY] [{ NULL | ERROR | DEFAULT <defaultExpr> } ON ERROR])	从JSON字符串中提取标量。
	此方法在JSON字符串中搜索给定的路径表达式，如果该路径上的值是标量，则返回该值。如果不是标量值，则无法返回。默认情况下，该值以STRING类型返回。使用returnType可以选择不同的类型，支持以下类型：
	VARCHAR / STRING
	BOOLEAN
	INTEGER
	DOUBLE
	对于空路径表达式或错误，可以定义为返回null、报错或返回定义的默认值。省略时，默认值为NULL ON EMPTY或NULL ON ERROR。默认值可以是字面量或表达式。如果默认值本身引发错误，那么它将执行ON EMPTY和ON ERROR的错误行为。
	`-- "true"`
	`JSON_VALUE('{"a": true}', '$.a')`
	
	`-- TRUE`
	`JSON_VALUE('{"a": true}', '$.a' RETURNING BOOLEAN)`
	
	`-- "false"`
	`JSON_VALUE('{"a": true}', 'lax $.b'`
	`DEFAULT FALSE ON EMPTY)`
	
	`-- "false"`
	`JSON_VALUE('{"a": true}', 'strict $.b'`
	`DEFAULT FALSE ON ERROR)`
	
	`-- 0.998D`
	`JSON_VALUE('{"a.b": [0.998,0.996]}','$.["a.b"][0]'`
	`RETURNING DOUBLE)`

 版权所有 © 华为技术有限公司

SQL函数	描述
JSON_QUERY(jsonValue, path [{ WITHOUT | WITH CONDITIONAL | WITH UNCONDITIONAL } [ARRAY] WRAPPER] [{ NULL | EMPTY ARRAY | EMPTY OBJECT | ERROR } ON EMPTY] [{ NULL | EMPTY ARRAY | EMPTY OBJECT | ERROR } ON ERROR])	从JSON字符串中提取JSON值。 结果始终以STRING形式返回。目前不支持RETURNING子句。 wrappingBehavior确定是否应该将提取的值包装到数组中，以及是无条件地还是只有当值本身不是数组时才这样做。 onEmpty和onError分别确定路径表达式为空或引发错误时的行为。默认情况下，在这两种情况下都返回null。其他选择是使用空数组、空对象或引发错误。 -- '{ "b": 1 }' JSON_QUERY('{ "a": { "b": 1 } }', '$.a') -- '[1, 2]' JSON_QUERY('[1, 2]', '$') -- NULL JSON_QUERY(CAST(NULL AS STRING), '$') -- '["c1","c2"]' JSON_QUERY('{"a":[{"c":"c1"},{"c":"c2"}]}', 'lax $.a[*].c') -- Wrap result into an array -- '[{}]' JSON_QUERY('{}', '$' WITH CONDITIONAL ARRAY WRAPPER) -- '[1, 2]' JSON_QUERY('[1, 2]', '$' WITH CONDITIONAL ARRAY WRAPPER) -- '[[1, 2]]' JSON_QUERY('[1, 2]', '$' WITH UNCONDITIONAL ARRAY WRAPPER) -- Scalars must be wrapped to be returned -- NULL JSON_QUERY(1, '$') -- '[1]' JSON_QUERY(1, '$' WITH CONDITIONAL ARRAY WRAPPER) -- Behavior if path expression is empty / there is an error -- '{}' JSON_QUERY('{}', 'lax $.invalid' EMPTY OBJECT ON EMPTY) -- '[]' JSON_QUERY('{}', 'strict $.invalid' EMPTY ARRAY ON ERROR)

版权所有 © 华为技术有限公司

SQL函数	描述

JSON_OBJECT([[KEY] key VALUE value]* [{ NULL | ABSENT } ON NULL])	从键值对列表构建JSON对象字符串。
	请注意，键必须是非NULL字符串文字，而值可以是任意表达式。
	函数返回一个JSON字符串。ON NULL行为定义了如何处理NULL值。如果省略，则默认为NULL ON NULL。
	从另一个JSON构造函数调用（JSON_OBJECT,JSON_ARRAY）创建的值将直接插入，而不是作为字符串插入。这允许构建嵌套的JSON结构。
	`-- '{}'`
	`JSON_OBJECT()`
	
	`-- '{"K1":"V1","K2":"V2"}'`
	`JSON_OBJECT('K1' VALUE 'V1', 'K2' VALUE 'V2')`
	
	`-- Expressions as values`
	`JSON_OBJECT('orderNo' VALUE orders.orderId)`
	
	`-- ON NULL`
	`JSON_OBJECT(KEY 'K1' VALUE CAST(NULL AS STRING) NULL ON NULL)  --`
	`'{"K1":null}'`
	`JSON_OBJECT(KEY 'K1' VALUE CAST(NULL AS STRING) ABSENT ON NULL) -- '{}'`
	
	`-- '{"K1":{"K2":"V"}}'`
	`JSON_OBJECT(`
	`  KEY 'K1'`
	`  VALUE JSON_OBJECT(`
	`    KEY 'K2'`
	`    VALUE 'V'`
	`  )`
	`)`

JSON_OBJECTAGG([KEY] key VALUE value [{ NULL | ABSENT } ON NULL])	通过将键值表达式聚合到单个JSON对象中来构建JSON对象字符串。
	键表达式必须返回一个不可为空的字符串。值表达式可以是任意的，包括其他JSON函数。如果值为NULL，则ON NULL行为定义要执行的操作。如果省略，则默认为NULL ON NULL。
	注意key必须是唯一的。如果一个key出现多次，则会抛出错误。
	目前在OVER窗口中不支持此功能。
	`-- '{"Apple":2,"Banana":17,"Orange":0}'`
	`SELECT`
	`  JSON_OBJECTAGG(KEY product VALUE cnt)`
	`FROM orders`

版权所有 © 华为技术有限公司

SQL函数	描述

JSON_ARRAY([value]* [{ NULL | ABSENT } ON NULL])	从值列表构建JSON数组字符串。
	该函数返回一个JSON字符串。这些值可以是任意表达式。ON NULL行为定义了如何处理NULL值。如果省略，则默认为ABSENT ON NULL。
	从另一个JSON构造函数调用（JSON_OBJECT, JSON_ARRAY）创建的元素将直接插入，而不是作为字符串插入。这允许构建嵌套的JSON结构。
	`-- '[]'`
	`JSON_ARRAY()`
	`-- '[1,"2"]'`
	`JSON_ARRAY(1, '2')`
	
	`-- Expressions as values`
	`JSON_ARRAY(orders.orderId)`
	
	`-- ON NULL`
	`JSON_ARRAY(CAST(NULL AS STRING) NULL ON NULL) -- '[null]'`
	`JSON_ARRAY(CAST(NULL AS STRING) ABSENT ON NULL) -- '[]'`
	
	`-- '[[1]]'`
	`JSON_ARRAY(JSON_ARRAY(1))`

JSON_ARRAYAGG(items [{ NULL | ABSENT } ON NULL])	通过将元素聚合到数组中来构建JSON对象字符串。
	元素表达式可以是任意的，包括其他JSON函数。如果值为NULL，则ON NULL行为定义要执行的操作。如果省略，则默认为ABSENT ON NULL。
	目前在OVER窗口、无界session窗口或hop窗口中不支持此功能。
	`-- '["Apple","Banana","Orange"]'`
	`SELECT`
	`  JSON_ARRAYAGG(product)`
	`FROM orders`

2.7.4.10 值构建函数

函数说明

表 2-91 值构建函数说明

值构建函数	函数说明
-- implicit constructor with parenthesis (value1 [, value2]*)	返回从值列表 (value1, value2, …) 创建的行。隐式行构造函数支持任意表达式作为字段，但至少需要两个字段。 显式行构造函数可以处理任意数量的字段，但目前还不能很好地支持所有类型的字段表达式。
ARRAY ‘[’ value1 [, value2]* ‘]’	返回从值列表 (value1, value2, …) 创建的数组。
MAP ‘[’ value1, value2 [, value3, value4]* ‘]’	返回从键值对列表 ((value1, value2), (value3, value4), …) 创建的 map。

版权所有 © 华为技术有限公司

2.7.4.11 值获取函数

表 2-92 值获取函数

SQL函数	描述
tableName.compositeType.field	按名称从 Flink 复合类型（例如，Tuple，POJO）返回字段的值。
tableName.compositeType.*	返回 Flink 复合类型（例如，Tuple，POJO）的平面表示，将其每个直接子类型转换为单独的字段。在大多数情况下，平面表示 的字段与原始字段的命名类似，但使用 $ 分隔符（例如 mypojo $mytuple$f0）。

2.7.4.12 分组函数

表 2-93 分组函数

SQL函数	函数说明
GROUP_ID()	返回唯一标识分组键组合的整数。
GROUPING(expression1 [, expression2]*) | GROUPING_ID(expression1 [, expression2]*)	返回给定分组表达式的位向量。

2.7.4.13 Hash 函数

表 2-94 Hash 函数

Hash函数	函数说明
MD5(string)	以 32 个十六进制数字的字符串形式返回 string 的 MD5 哈希值；如果字符串为 NULL，则返回 NULL。
SHA1(string)	以 40 个十六进制数字的字符串形式返回 string 的 SHA-1 哈希值；如果字符串为 NULL，则返回 NULL。
SHA224(string)	以 56 个十六进制数字的字符串形式返回 string 的 SHA-224 哈希值；如果字符串为 NULL，则返回 NULL。
SHA256(string)	以 64 个十六进制数字的字符串形式返回 string 的 SHA-256 哈希值；如果字符串为 NULL，则返回 NULL。

版权所有 © 华为技术有限公司

Hash函数	函数说明
SHA384(string)	以 96 个十六进制数字的字符串形式返回 string 的 SHA-384 哈希值；如果字符串为 NULL，则返回 NULL。
SHA512(string)	以 128 个十六进制数字的字符串形式返回 string 的 SHA-512 哈希值；如果字符串为 NULL，则返回 NULL。
SHA2(string, hashLength)	使用 SHA-2 系列散列函数（SHA-224，SHA-256，SHA-384 或 SHA-512）返回散列值。第一个参数字符串是要散列的字符串， 第二个参数 hashLength 是结果的位长（224，256，384 或 512）。如果 string 或 hashLength 为 NULL，则返回 NULL。

2.7.4.14 聚合函数

聚合函数将所有的行作为输入，并返回单个聚合值作为结果。

表 2-95 聚合函数

函数	描述
COUNT([ALL] expression | DISTINCT expression1 [, expression2]*)	默认情况下或使用关键字 ALL，返回不为 NULL 的表达式的输入行数。使用 DISTINCT 则对所有值去重后计算。
COUNT(*) | COUNT(1)	返回输入行数。
AVG([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行中表达式的平均值（算术平均值）。使用 DISTINCT 则对所有值去重后计算。
SUM([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行的表达式总和。使用 DISTINCT 则对所有值去重后计算。
MAX([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行中表达式的最大值。使用 DISTINCT 则对所有值去重后计算。
MIN([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行中表达式的最小值。使用 DISTINCT 则对所有值去重后计算。
STDDEV_POP([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行中表达式的总体标准偏差。使用 DISTINCT 则对所有值去重后计算。
STDDEV_SAMP([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行中表达式的样本标准偏差。使用 DISTINCT 则对所有值去重后计算。

版权所有 © 华为技术有限公司

函数	描述
VAR_POP([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行中表达式的总体方差（总体标准差的平方）。使用 DISTINCT 则对所有值去重后计算。
VAR_SAMP([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回所有输入行中表达式的样本方差（样本标准差的平方）。使用 DISTINCT 则对所有值去重后计算。
COLLECT([ALL | DISTINCT] expression)	默认情况下或使用关键字 ALL，返回跨所有输入行的多组表达式。NULL 值将被忽略。使用 DISTINCT 则对所有值去重后计算
VARIANCE([ALL | DISTINCT] expression)	VAR_SAMP() 的同义方法。
RANK()	返回值在一组值中的排名。结果是 1 加上分区顺序中当前行之前或等于当前行的行数。排名在序列中不一定连续。
DENSE_RANK()	返回值在一组值中的排名。结果是一加先前分配的等级值。与函数 rank 不同，dense_rank 不会在排名序列中产生间隙。
ROW_NUMBER()	在窗口分区内根据 rows 的排序为每一行分配一个唯一的序列号，从一开始。ROW_NUMBER 和 RANK 相似。ROW_NUMBER 按 顺序对所有行进行编号（例如 1，2，3，4，5）。RANK 为等值 row 提供相同的序列值（例如 1，2，2，4，5）。
LEAD(expression [, offset] [, default])	返回窗口中当前行之后第 offset 行处的表达式值。offset 的默认值为 1，default 的默认值为 NULL。
LAG(expression [, offset] [, default])	返回窗口中当前行之前第 offset 行处的表达式值。offset 的默认值为 1，default 的默认值为 NULL。
FIRST_VALUE(expression)	返回一组有序值中的第一个值。
LAST_VALUE(expression)	返回一组有序值中的最后一个值。
LISTAGG(expression [, separator])	连接字符串表达式的值并在它们之间放置分隔符值。字符串末尾不添加分隔符时则分隔符的默认值为“,”。

2.7.4.15 表值函数

2.7.4.15.1 string_split

string_split函数，根据指定的分隔符将目标字符串拆分为子字符串，并返回子字符串列表。

语法说明

```
string_split(target, separator)
```


版权所有 © 华为技术有限公司

表 2-96 string_split 参数说明

参数	数据类型	说明
target	STRING	待处理的目标字符串。 **说明** • 如果target为NULL，则返回一个空行。 • 如果target包含两个或多个连续出现的分隔符时，则返回长度为零的空子字符串。 • 如果target未包含指定分隔符，则返回目标字符串。
separator	VARCHAR	指定的分隔符，当前仅支持单字符分隔。

示例

1. 参考Kafka和Print创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.15”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   CREATE TABLE kafkaSource (
     target STRING,
     separator  VARCHAR
   ) WITH (
     'connector' = 'kafka',
     'topic' = '***KafkaTopic***',
     'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
     'properties.group.id' = '***GroupId***',
     'scan.startup.mode' = 'latest-offset',
     'format' = 'json'
   );

   CREATE TABLE printSink (
     target STRING,
     item STRING
   ) WITH (
     'connector' = 'print'
   );
   insert into printSink select target,  item from kafkaSource, lateral table(string_split(target, separator))
   as T(item);
   ```

2. 连接Kafka集群，向Kafka的topic中发送如下测试数据：

   ```
   {"target":"test-flink","separator":"-"}
   {"target":"flink","separator":"-"}
   {"target":"one-two-ww-three","separator":"-"}
   ```

   即数据如下：

版权所有 © 华为技术有限公司

表 2-97 测试源表数据和分隔符

target（STRING）	separator （VARCHAR）
test-flink	-
flink	-
one-two-ww-three	-

3. 查看输出结果。
   - 方法一：
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
     iii. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
   - 方法二：如果在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
     iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

查询结果参考如下：

```
+I(test-flink,test)
+I(test-flink,flink)
+I(flink,flink)
+I(one-two-ww-three,one)
+I(one-two-ww-three,two)
+I(one-two-ww-three,ww)
+I(one-two-ww-three,three)
```

即数据输出结果参考如下：

表 2-98 结果表数据

target（STRING）	item（STRING）
test-flink	test
test-flink	flink
flink	flink
one-two-ww-three	one
one-two-ww-three	two
one-two-ww-three	ww
one-two-ww-three	three

版权所有 © 华为技术有限公司

3 Flink Opensource SQL1.12 语法参考

3.1 SQL 语法约束与定义

3.1.1 语法支持类型

DLI SQL语法支持以下数据类型：

STRING，BOOLEAN，BYTES，DECIMAL，TINYINT，SMALLINT，INTEGER，BIGINT，FLOAT，DOUBLE，DATE，TIME，TIMESTAMP，TIMESTAMP WITH LOCAL TIME ZONE，INTERVAL，ARRAY，MULTISET，MAP，ROW

在SQL语法中这些类型用于定义表中列的数据类型。

3.1.2 语法定义

3.1.2.1 DDL 语法定义

3.1.2.1.1 CREATE TABLE 语句

语法定义

```
CREATE TABLE table_name
  (
    { <column_definition> | <computed_column_definition> }[ , ...n]
    [ <watermark_definition> ]
    [ <table_constraint> ][ , ...n]
  )
  [COMMENT table_comment]
  [PARTITIONED BY (partition_column_name1, partition_column_name2, ...)]
  WITH (key1=val1, key2=val2, ...)

<column_definition>:
  column_name column_type [ <column_constraint> ] [COMMENT column_comment]

<column_constraint>:
  [CONSTRAINT constraint_name] PRIMARY KEY NOT ENFORCED

<table_constraint>:
  [CONSTRAINT constraint_name] PRIMARY KEY (column_name, ...) NOT ENFORCED
```

 版权所有 © 华为技术有限公司

```
<computed_column_definition>:
  column_name AS computed_column_expression [COMMENT column_comment]

<watermark_definition>:
  WATERMARK FOR rowtime_column_name AS watermark_strategy_expression

<source_table>:
  [catalog_name.][db_name.]table_name
```

功能描述

根据指定的表名创建一个表。

语法说明

COMPUTED COLUMN

计算列是一个使用 “column_name AS computed_column_expression” 语法生成的虚拟列。它由使用同一表中其他列的非查询表达式生成，并且不会在表中进行物理存储。例如，一个计算列可以使用 cost AS price * quantity 进行定义，这个表达式可以包含物理列、常量、函数或变量的任意组合，但这个表达式不能存在任何子查询。

在 Flink 中计算列一般用于为 CREATE TABLE 语句定义 时间属性。 处理时间属性 可以简单地通过使用了系统函数 PROCTIME() 的 proc AS PROCTIME() 语句进行定义。另一方面，由于事件时间列可能需要从现有的字段中获得，因此计算列可用于获得事件时间列。例如，原始字段的类型不是 TIMESTAMP(3) 或嵌套在 JSON 字符串中。

注意：

- 定义在一个数据源表（ source table ）上的计算列会在从数据源读取数据后被计算，它们可以在 SELECT 查询语句中使用。
- 计算列不可以作为 INSERT 语句的目标，在 INSERT 语句中，SELECT 语句的 schema 需要与目标表不带有计算列的 schema 一致。

WATERMARK

WATERMARK 定义了表的事件时间属性，其形式为 WATERMARK FOR rowtime_column_name AS watermark_strategy_expression 。

rowtime_column_name 把一个现有的列定义为一个为表标记事件时间的属性。该列的类型必须为 TIMESTAMP(3)，且是 schema 中的顶层列，它也可以是一个计算列。

watermark_strategy_expression 定义了 watermark 的生成策略。它允许使用包括计算列在内的任意非查询表达式来计算 watermark ； 表达式的返回类型必须是 TIMESTAMP(3)，表示了从 Epoch 以来的经过的时间。 返回的 watermark 只有当其不为空且其值大于之前发出的本地 watermark 时才会被发出（以保证 watermark 递增）。每条记录的 watermark 生成表达式计算都会由框架完成。 框架会定期发出所生成的最大的 watermark ，如果当前 watermark 仍然与前一个 watermark 相同、为空、或返回的 watermark 的值小于最后一个发出的 watermark ，则新的 watermark 不会被发出。 Watermark 根据 pipeline.auto-watermark-interval 中所配置的间隔发出。 若 watermark 的间隔是 0ms ，那么每条记录都会产生一个 watermark，且 watermark 会在不为空并大于上一个发出的 watermark 时发出。

使用事件时间语义时，表必须包含事件时间属性和 watermark 策略。

Flink 提供了几种常用的 watermark 策略。

 版权所有 © 华为技术有限公司

- 严格递增时间戳： WATERMARK FOR rowtime_column AS rowtime_column。

  发出到目前为止已观察到的最大时间戳的 watermark ，时间戳大于最大时间戳的行被认为没有迟到。

- 递增时间戳： WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '0.001' SECOND。

  发出到目前为止已观察到的最大时间戳减 1 的 watermark ，时间戳大于或等于最大时间戳的行被认为没有迟到。

- 有界乱序时间戳： WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL 'string' timeUnit。

  发出到目前为止已观察到的最大时间戳减去指定延迟的 watermark ，例如，WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '5' SECOND 是一个 5 秒延迟的 watermark 策略。

```
CREATE TABLE Orders (
    user BIGINT,
    product STRING,
    order_time TIMESTAMP(3),
    WATERMARK FOR order_time AS order_time - INTERVAL '5' SECOND
) WITH ( . . . );
```

PRIMARY KEY

主键用作 Flink 优化的一种提示信息。主键限制表明一张表或视图的某个（些）列是唯一的并且不包含 Null 值。 主键声明的列都是非 nullable 的。因此主键可以被用作表行级别的唯一标识。

主键可以和列的定义一起声明，也可以独立声明为表的限制属性，不管是哪种方式，主键都不可以重复定义，否则 Flink 会报错。

有效性检查

SQL 标准主键限制可以有两种模式：ENFORCED 或者 NOT ENFORCED。 它申明了是否输入/出数据会做合法性检查（是否唯一）。Flink 不存储数据因此只支持 NOT ENFORCED 模式，即不做检查，用户需要自己保证唯一性。

Flink 假设声明了主键的列都是不包含 Null 值的，Connector 在处理数据时需要自己保证语义正确。

注意: 在 CREATE TABLE 语句中，创建主键会修改列的 nullable 属性，主键声明的列默认都是非 Nullable 的。

PARTITIONED BY

根据指定的列对已经创建的表进行分区。若表使用 filesystem sink ，则将会为每个分区创建一个目录。

WITH OPTIONS

表属性用于创建 table source/sink ，一般用于寻找和创建底层的连接器。

表达式 key1=val1 的键和值必须为字符串文本常量。

注意： 使用 CREATE TABLE 语句注册的表均可用作 table source 和 table sink。 在被 DML 语句引用前，我们无法决定其实际用于 source 抑或是 sink。

 版权所有 © 华为技术有限公司

3.1.2.1.2 CREATE VIEW 语句

语法定义

```
CREATE VIEW [IF NOT EXISTS] view_name
  [{columnName [, columnName ]* }] [COMMENT view_comment]
  AS query_expression
```

功能描述

通过定义数据视图的方式，将多层嵌套写在数据视图中，简化开发过程。

语法说明

IF NOT EXISTS

若该视图已经存在，则不会进行任何操作。

示例

创建一个名为viewName的视图

```
create view viewName as select * from dataSource
```

3.1.2.1.3 CREATE FUNCTION 语句

语法定义

```
CREATE FUNCTION
  [IF NOT EXISTS] function_name
  AS identifier [LANGUAGE JAVA|SCALA]
```

功能描述

创建一个用户自定义函数。

如果您需要了解创建自定义函数的步骤请参考**自定义函数**。

语法说明

IF NOT EXISTS

若该函数已经存在，则不会进行任何操作。

LANGUAGE JAVA|SCALA

Language tag 用于指定 Flink runtime 如何执行这个函数。目前，只支持 JAVA 和 SCALA，且函数的默认语言为 JAVA。

示例

创建一个名为STRINGBACK的函数

```
create function STRINGBACK as 'com.dli.StringBack'
```

 版权所有 © 华为技术有限公司

3.1.2.2 DML 语法定义

DML 语句

语法定义

```
INSERT INTO table_name [PARTITION part_spec] query

part_spec:  (part_col_name1=val1 [, part_col_name2=val2, ...])

query:
  values
  | {
      select
      | selectWithoutFrom
      | query UNION [ ALL ] query
      | query EXCEPT query
      | query INTERSECT query
    }
    [ ORDER BY orderItem [, orderItem ]* ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start { ROW | ROWS } ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]

orderItem:
  expression [ ASC | DESC ]

select:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
  [ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]

selectWithoutFrom:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }

projectItem:
  expression [ [ AS ] columnAlias ]
  | tableAlias . *

tableExpression:
  tableReference [, tableReference ]*
  | tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ] JOIN tableExpression [ joinCondition ]

joinCondition:
  ON booleanExpression
  | USING '(' column [, column ]* ')'

tableReference:
  tablePrimary
  [ matchRecognize ]
  [ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ] ]

tablePrimary:
  [ TABLE ] [ [ catalogName . ] schemaName . ] tableName
  | LATERAL TABLE '(' functionName '(' expression [, expression ]* ')' ')'
  | UNNEST '(' expression ')'

values:
  VALUES expression [, expression ]*

groupItem:
  expression
  | '(' ')'
```

 版权所有 © 华为技术有限公司

```
  | '(' expression [, expression ]* ')'
  | CUBE '(' expression [, expression ]* ')'
  | ROLLUP '(' expression [, expression ]* ')'
  | GROUPING SETS '(' groupItem [, groupItem ]* ')'

windowRef:
    windowName
  | windowSpec

windowSpec:
    [ windowName ]
    '('
    [ ORDER BY orderItem [, orderItem ]* ]
    [ PARTITION BY expression [, expression ]* ]
    [
        RANGE numericOrIntervalExpression {PRECEDING}
      | ROWS numericExpression {PRECEDING}
    ]
    ')'

matchRecognize:
      MATCH_RECOGNIZE '('
      [ PARTITION BY expression [, expression ]* ]
      [ ORDER BY orderItem [, orderItem ]* ]
      [ MEASURES measureColumn [, measureColumn ]* ]
      [ ONE ROW PER MATCH ]
      [ AFTER MATCH
            ( SKIP TO NEXT ROW
            | SKIP PAST LAST ROW
            | SKIP TO FIRST variable
            | SKIP TO LAST variable
            | SKIP TO variable )
      ]
      PATTERN '(' pattern ')'
      [ WITHIN intervalLiteral ]
      DEFINE variable AS condition [, variable AS condition ]*
      ')'

measureColumn:
      expression AS alias

pattern:
      patternTerm [ '|' patternTerm ]*

patternTerm:
      patternFactor [ patternFactor ]*

patternFactor:
      variable [ patternQuantifier ]

patternQuantifier:
      '*'
  |   '*?'
  |   '+'
  |   '+?'
  |   '?'
  |   '??'
  |   '{' { [ minRepeat ], [ maxRepeat ] } '}' ['?']
  |   '{' repeat '}'
```

注意事项

Flink SQL 对于标识符（表、属性、函数名）有类似于 Java 的词法约定:

- 不管是否引用标识符，都保留标识符的大小写。
- 且标识符需区分大小写。
- 与 Java 不一样的地方在于，通过反引号，可以允许标识符带有非字母的字符（如："SELECT a AS `my field` FROM t"）。

字符串文本常量需要被单引号包起来（如 SELECT 'Hello World' ）。两个单引号表示转义（如 SELECT 'It''s me.' ）。字符串文本常量支持 Unicode 字符，如需明确使用 Unicode 编码，请使用以下语法：

- 使用反斜杠（\）作为转义字符（默认）： SELECT U&'\263A'
- 使用自定义的转义字符： SELECT U&'#263A' UESCAPE '#'

3.2 Flink OpenSource SQL1.12 语法概览

本章节介绍目前DLI所提供的Flink OpenSource SQL1.12语法列表。参数说明，示例等详细信息请参考具体的语法说明。

创建表相关语法

表 3-1 创建表相关语法

语法分类	功能描述
创建源表	DataGen源表
	DWS源表
	Hbase源表
	JDBC源表
	Kafka源表
	MySQL CDC源表
	Postgres CDC源表
	Redis源表
	Upsert Kafka源表
创建结果表	BlackHole结果表
	ClickHouse结果表
	DWS结果表
	Elasticsearch结果表
	Hbase结果表
	JDBC结果表
	Kafka结果表
	Print结果表
	Redis结果表
	Upsert Kafka结果表
创建维表	DWS维表

版权所有 © 华为技术有限公司

语法分类	功能描述
	Hbase维表
	JDBC维表
	Redis维表
Format	Avro
	Canal
	Confluent Avro
	CSV
	Debezium
	JSON
	Maxwell
	Raw

3.3 数据定义语句 DDL

3.3.1 创建源表

3.3.1.1 DataGen 源表

功能描述

DataGen主要用于生成随机数据，可用于调试以及测试等场景。

前提条件

无

注意事项

- 创建DataGen表时，表字段类型不支持Array，Map和Row复杂类型，可以通过CREATE TABLE语句中的“COMPUTED COLUMN”来进行类似功能构造。
- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table dataGenSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)
```

 版权所有 © 华为技术有限公司

```
)
with (
  'connector' = 'datagen'
);
```

参数说明

表 3-2 参数说明

参数	是否必选	默认值	数据类型	参数说明
connector	是	无	String	指定要使用的连接器，这里是'datagen'。
rows-per-second	否	10000	Long	每秒生成的行数，用以控制数据发出速率。
fields.#.kind	否	random	String	指定 '#' 字段的生成器。 **'#' 字段必须是DataGen表中的字段，实际使用时需要将'#'替换为相应字段名。**其他各参数的'#'号意义相同，不再重复描述。 参数值可以是 'sequence' 或 'random'，具体含义如下： ● random是默认的生成器，您可以通过“fields.#.max”和“fields.#.min”参数指定随机生成的最大和最小值。当指定的字段类型为char、varchar、string时，可以同时通过“fields.#.length”字段指定长度。random是无界的生成器。 ● sequence生成器，您可以通过“fields.#.start”和“fields.#.end”指定序列的起始和结束值。sequence是有界的生成器，当序列数字达到结束值，读取结束。
fields.#.min	否	'#'号指定的字段类型的最小值	'#'号指定的字段类型	当“fields.#.kind”字段为：random时有效。 表示随机生成器的最小值，'#' 指定的字段仅适用于数字类型。
fields.#.max	否	'#'号指定的字段类型的最大值	'#'号指定的字段类型	当“fields.#.kind”字段为：random时有效。 随机生成器的最大值，'#' 指定的字段仅适用于数字类型。
fields.#.length	否	100	Integer	当“fields.#.kind”字段为：random时有效。 随机生成器生成字符的长度，#' 指定的字段仅适用于char、varchar、string。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
fields.#.start	否	无	'#'号指定的字段类型	当“fields.#.kind”字段为：sequence时有效。 序列生成器的起始值。
fields.#.end	否	无	'#'号指定的字段类型	当“fields.#.kind”字段为：sequence时有效。 序列生成器的结束值。

示例

参考**创建Flink OpenSource作业**，创建flink opensource sql作业，运行如下作业脚本，通过DataGen表产生随机数据并输出到Print结果表中。

注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

```
create table dataGenSource(
  user_id string,
  amount int
) with (
  'connector' = 'datagen',
  'rows-per-second' = '1', --每秒生成一条数据
  'fields.user_id.kind' = 'random', --为字段user_id指定random生成器
  'fields.user_id.length' = '3' --限制user_id长度为3
);

create table printSink(
  user_id string,
  amount int
) with (
  'connector' = 'print'
);

insert into printSink select * from dataGenSource;
```

该作业提交后，作业状态变成“运行中”，后续您可通过如下操作查看输出结果。

- 方法一：
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
  c. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
- 方法二：若在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
  c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

版权所有 © 华为技术有限公司

3.3.1.2 DWS 源表

功能描述

DLI将Flink作业从数据仓库服务（DWS）中读取数据。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见**《数据仓库服务管理指南》**。

前提条件

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。
  如何创建DWS集群，请参考《数据仓库服务管理指南》中“创建集群”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。
  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector' = 'gaussdb',
  'url' = '',
  'table-name' = '',
  'username' = '',
  'password' = ''
);
```


版权所有 © 华为技术有限公司

参数说明

表 3-3 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'gaussdb'。
url	是	无	String	jdbc连接地址。“url”参数中的ip地址请使用DWS的内网地址。 使用gsjdbc4驱动连接时，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。 使用gsjdbc200驱动连接时，格式为：jdbc:gaussdb://${ip}:${port}/${dbName}。
table-name	是	无	String	操作的DWS表名。如果该DWS表在某schema下，则具体可以参考**如果该DWS表在某schema下**的说明。
driver	否	org.postgresql.Driver	String	jdbc连接驱动，默认为: org.postgresql.Driver。 • 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。 • 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。
username	否	无	String	DWS数据库认证用户名，需要和'password'参数一起配置。
password	否	无	String	DWS数据库认证密码，需要和'username'参数一起配置。
scan.partition.column	否	无	String	用于对输入进行分区的列名。 注意：该参数与scan.partition.lower-bound、scan.partition.upper-bound、 scan.partition.num参数必须同时配置或者同时都不配置。
scan.partition.lower-bound	否	无	Integer	第一个分区的最小值。 与scan.partition.column、scan.partition.upper-bound、 scan.partition.num必须同时配置或者同时都不配置。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
scan.partition.upper-bound	否	无	Integer	最后一个分区的最大值。 与scan.partition.column、scan.partition.lower-bound、 scan.partition.num必须同时配置或者同时都不配置。
scan.partition.num	否	无	Integer	分区的个数。 与scan.partition.column、scan.partition.upper-bound、 scan.partition.upper-bound必须同时配置或者同时都不配置。
scan.fetch-size	否	0	Integer	每次从数据库拉取数据的行数。默认值为0，表示不限制。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置账号和密码。

示例

该示例是从DWS数据源中读取数据，并写入到Print结果表中，其具体步骤参考如下：

1. 在DWS中创建相应的表，表名为dws_order，SQL语句参考如下。

```
create table public.dws_order(
  order_id VARCHAR,
  order_channel VARCHAR,
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR);
```

   在DWS中执行以下SQL语句，向dws_order表中插入数据。

```
insert into public.dws_order
  (order_id,
  order_channel,
  order_time,
  pay_amount,
  real_pay,
  pay_time,
  user_id,
  user_name,
  area_id) values
  ('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',
'0001', 'Alice', '330106'),
  ('20210325120202001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',
'0002', 'Bob', '330110');
```

2. 参考**增强型跨源连接**，根据DWS所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。

版权所有 © 华为技术有限公司

3. 设置DWS的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据DWS的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将DWS作为数据源，Print作为结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE dwsSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DWSIP:DWSPort/DWSdbName',
  'table-name' = 'dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword'
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from dwsSource;
```

5. 按照如下操作查看taskmanager.out文件中的数据结果。
   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

```
+I(20210324100000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24
10:02:03,0001,Alice,330106)
+I(20210325120200001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25
12:03:00,0002,Bob,330110)
```

常见问题

- Q：作业运行失败，运行日志中有如下报错信息，应该怎么解决?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
```

 版权所有 © 华为技术有限公司

```
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

A：应考虑是跨源没有绑定，或者跨源没有绑定成功。

- 参考**增强型跨源连接**章节，重新配置跨源。参考**DLI跨源连接DWS失败**进行问题排查。

- Q：如果该DWS表在某schema下，应该如何配置?

  A：如下示例是使用schema为dbuser2下的表dws_order。

```
CREATE TABLE dwsSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DWSIP:DWSPort/DWSdbName',
  'table-name' = 'dbuser2\".\"dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword'
);
```

3.3.1.3 Hbase 源表

功能描述

创建source流从HBase中获取数据，作为作业的输入数据。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。DLI可以从HBase中读取数据，用于过滤分析、数据转储等场景。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- 若使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机IP信息。

  详细操作请参考《数据湖探索用户指南》中的“**修改主机信息**”章节描述。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 创建HBase源表的列簇必须定义为ROW类型，字段名对应列簇名（column family），嵌套的字段名对应列限定符名（column qualifier）。

  用户只需在表结构中声明查询中使用的的列簇和列限定符。除了ROW类型的列，剩下的原子数据类型字段（比如，STRING, BIGINT）将被识别为HBase的rowkey，一张表中只能声明一个rowkey。rowkey字段的名字可以是任意的，如果是保留关键字，需要用反引号进行转义。

语法格式

```
create table hbaseSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
  ','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'hbase-2.2',
  'table-name' = '',
  'zookeeper.quorum' = ''
);
```

参数说明

表 3-4 参数说明

参数	是否必选	默认值	数据	说明
connector	是	无	String	指定使用的连接器，需配置为：hbase-2.2。
table-name	是	无	String	连接的HBase表名。
zookeeper.quorum	是	无	String	格式为：ZookeeperAddress:ZookeeperPort 以MRS Hbase集群为例，该参数的所使用Zookeeper的ip地址和端口号获取方式如下： ● 在MRS Manager上，选择“集群 > *待操作的集群名称* > 服务 > ZooKeeper > 实例”，获取ZooKeeper角色实例的IP地址。 ● 在MRS Manager上，选择“集群 > 待操作的集群名称 > 服务 > ZooKeeper > 配置 > 全部配置”，搜索参数“clientPort”，获取“clientPort”的参数值即为ZooKeeper的端口。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据	说明
zookeeper.znode.parent	否	/hbase	String	Zookeeper中的根目录，默认是/hbase。
null-string-literal	否	无	String	当字符串值为null时的存储形式，默认存成"null" 字符串。 HBase的source的编解码将所有数据类型（除字符串外）将null值以空字节来存储。
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。

数据类型映射

HBase以字节数组存储所有数据，在读和写过程中要序列化和反序列化数据。

Flink的HBase连接器利用HBase（Hadoop) 的工具类 org.apache.hadoop.hbase.util.Bytes进行字节数组和Flink数据类型转换。

Flink的HBase连接器将所有数据类型（除字符串外）null值编码成空字节。对于字符串类型，null值的字面值由null-string-literal选项值决定。

表 3-5 数据类型映射表

Flink数据类型	HBase转换
CHAR/VARCHAR/STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY/VARBINARY	返回 byte[]。
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
SMALLINT	byte[] toBytes(short val) short toShort(byte[] bytes)
INT	byte[] toBytes(int val) int toInt(byte[] bytes)
BIGINT	byte[] toBytes(long val) long toLong(byte[] bytes)

版权所有 © 华为技术有限公司

Flink数据类型	HBase转换
FLOAT	byte[] toBytes(float val) float toFloat(byte[] bytes)
DOUBLE	byte[] toBytes(double val) double toDouble(byte[] bytes)
DATE	从 1970-01-01 00:00:00 UTC 开始的天数，int 值。
TIME	从 1970-01-01 00:00:00 UTC 开始天的毫秒数，int 值。
TIMESTAMP	从 1970-01-01 00:00:00 UTC 开始的毫秒数，long 值。
ARRAY	不支持
MAP/MULTISET	不支持
ROW	不支持

示例

该示例是从HBase数据源中读取数据，并写入到Print结果表中，其具体步骤参考如下（该示例使用的HBase版本1.3.1和2.1.1和2.2.3）：

1. 参考**增强型跨源连接**，在DLI上根据HBase所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink作业队列。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置HBase集群的安全组，添加入向规则使其对Flink作业队列网段放通。参考**测试地址连通性**根据HBase的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**MRS HBase的使用**，通过HBase shell在HBase中创建相应的表，表名为order，表中只有一个列簇detail。创建语句参考如下：

   ```
   create 'order', {NAME => 'detail'}
   ```

4. 在HBase shell中执行下述命令，以插入一条数据：

   ```
   put 'order', '202103241000000001', 'detail:order_channel','webShop'
   put 'order', '202103241000000001', 'detail:order_time','2021-03-24 10:00:00'
   put 'order', '202103241000000001', 'detail:pay_amount','100.00'
   put 'order', '202103241000000001', 'detail:real_pay','100.00'
   put 'order', '202103241000000001', 'detail:pay_time','2021-03-24 10:02:03'
   put 'order', '202103241000000001', 'detail:user_id','0001'
   put 'order', '202103241000000001', 'detail:user_name','Alice'
   put 'order', '202103241000000001', 'detail:area_id','330106'
   ```

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将HBase作为数据源，Print作为结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   create table hbaseSource (
     order_id string,--表示唯一的rowkey
     detail Row( --detail表示列簇
       order_channel string,
   ```


版权所有 © 华为技术有限公司

```
  order_time string,
  pay_amount string,
  real_pay string,
  pay_time string,
  user_id string,
  user_name string,
  area_id string),
 primary key (order_id) not enforced
) with (
 'connector' = 'hbase-2.2',
  'table-name' = 'order',
  'zookeeper.quorum' = 'ZookeeperAddress:ZookeeperPort'
) ;

create table printSink (
 order_id string,
 order_channel string,
 order_time string,
 pay_amount string,
 real_pay string,
 pay_time string,
 user_id string,
 user_name string,
 area_id string
) with (
 'connector' = 'print'
);

insert into printSink select order_id,
detail.order_channel,detail.order_time,detail.pay_amount,detail.real_pay,
detail.pay_time,detail.user_id,detail.user_name,detail.area_id from hbaseSource;
```

6. 按照如下方式查看taskmanager.out文件中的数据结果：

   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

   ```
   +I(202103241000000001,webShop,2021-03-24 10:00:00,100.00,100.00,2021-03-24
   10:02:03,0001,Alice,330106)
   ```

常见问题

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决？

  ```
  java.lang.IllegalArgumentException: offset (0) + length (8) exceed the capacity of the array: 6
  ```

  A：如果HBase表中的数据是以其他方式导入的话，那么其存储是以String格式存储的，所以使用其他的数据格式将会报该错误。需要将Flink创建HBase源表中非string类型的字段的字段类型重新改为String即可。

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决？

  ```
  org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard
  from server in 90069ms for connection id 0x0
  ```

  A：跨源未绑定或未绑定成功，或是HBase集群安全组未配置放通DLI队列的网段地址。参考**增强型跨源连接**重新配置跨源，或者HBase集群安全组放通DLI队列的网段地址。

 版权所有 © 华为技术有限公司

3.3.1.4 JDBC 源表

功能描述

JDBC连接器是Flink内置的Connector，用于从数据库读取相应的数据。

前提条件

- 要与实例建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table jbdcSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
) with (
  'connector' = 'jdbc',
  'url' = '',
  'table-name' = '',
  'username' = '',
  'password' = ''
);
```

参数说明

表 3-6 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定要使用的连接器，当前固定为'jdbc'。
url	是	无	String	数据库的URL。
table-name	是	无	String	读取数据库中的数据所在的表名。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
driver	否	无	String	连接数据库所需要的驱动。若未配置，则会自动通过URL提取。
username	否	无	String	数据库认证用户名，需要和'password'一起配置。
password	否	无	String	数据库认证密码，需要和'username'一起配置。
scan.partition.column	否	无	String	用于对输入进行分区的列名。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.partition.num	否	无	Integer	分区的个数。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.partition.lower-bound	否	无	Integer	第一个分区的最小值。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.partition.upper-bound	否	无	Integer	最后一个分区的最大值。分区扫描参数，具体请参考**分区扫描功能介绍**。
scan.fetch-size	否	0	Integer	每次从数据库拉取数据的行数。若指定为0，则会忽略sql hint。
scan.auto-commit	否	true	Boolean	是否设置自动提交，以确定事务中的每个statement是否自动提交
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。用户若配置该配置项则不用在SQL中配置账号和密码。

分区扫描功能介绍

为了加速Source任务实例中的数据读取，Flink为JDBC表提供了分区扫描功能。以下参数定义了从多个任务并行读取时如何对表进行分区。

- scan.partition.column：用于对输入进行分区的列名，该列的数据类型必须是数字，日期或时间戳。
- scan.partition.num: 分区数。
- scan.partition.lower-bound：第一个分区的最小值。
- scan.partition.upper-bound：最后一个分区的最大值。

说明

- **建表时以上扫描分区参数必须同时存在或者同时不存在。**
- scan.partition.lower-bound和scan.partition.upper-bound参数仅用于决定分区步长，而不是用于过滤表中的行，表中的所有行都会被分区并返回。

版权所有 © 华为技术有限公司

数据类型映射

表 3-7 数据类型映射

MySQL类型	PostgreSQL类型	Flink SQL类型
TINYINT	-	TINYINT
SMALLINT TINYINT UNSIGNED	SMALLINT INT2 SMALLSERIAL SERIAL2	SMALLINT
INT MEDIUMINT SMALLINT UNSIGNED	INTEGER SERIAL	INT
BIGINT INT UNSIGNED	BIGINT BIGSERIAL	BIGINT
BIGINT UNSIGNED	-	DECIMAL(20, 0)
BIGINT	BIGINT	BIGINT
FLOAT	REAL FLOAT4	FLOAT
DOUBLE DOUBLE PRECISION	FLOAT8 DOUBLE PRECISION	DOUBLE
NUMERIC(p, s) DECIMAL(p, s)	NUMERIC(p, s) DECIMAL(p, s)	DECIMAL(p, s)
BOOLEAN TINYINT(1)	BOOLEAN	BOOLEAN
DATE	DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]

版权所有 © 华为技术有限公司

MySQL类型	PostgreSQL类型	Flink SQL类型
CHAR(n) VARCHAR(n) TEXT	CHAR(n) CHARACTER(n) VARCHAR(n) CHARACTER VARYING(n) TEXT	STRING
BINARY VARBINARY BLOB	BYTEA	BYTES
-	ARRAY	ARRAY

示例

使用JDBC作为数据源，Print作为sink，从RDS MySQL数据库中读取数据，并写入到Print中。

1. 参考**增强型跨源连接**，根据RDS MySQL所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置RDS MySQL的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根RDS的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 登录RDS MySQL，并使用下述命令在flink库下创建orders表，并插入数据。创建数据库的操作可以参考**创建RDS数据库**。

   在flink数据库库下创建orders表：

   ```
   CREATE TABLE `flink`.`orders` (
     `order_id` VARCHAR(32) NOT NULL,
     `order_channel` VARCHAR(32) NULL,
     `order_time` VARCHAR(32) NULL,
     `pay_amount` DOUBLE UNSIGNED NOT NULL,
     `real_pay` DOUBLE UNSIGNED NULL,
     `pay_time` VARCHAR(32) NULL,
     `user_id` VARCHAR(32) NULL,
     `user_name` VARCHAR(32) NULL,
     `area_id` VARCHAR(32) NULL,
     PRIMARY KEY (`order_id`)
   )    ENGINE = InnoDB
     DEFAULT CHARACTER SET = utf8mb4
     COLLATE = utf8mb4_general_ci;
   ```

   插入表数据：

   ```
   insert into orders(
     order_id,
     order_channel,
     order_time,
     pay_amount,
     real_pay,
     pay_time,
     user_id,
     user_name,
   ```


版权所有 © 华为技术有限公司

```
  area_id) values
  ('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',
'0001', 'Alice', '330106'),
  ('202103251202020001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',
'0002', 'Bob', '330110');
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE jdbcSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://MySQLAddress:MySQLPort/flink',--flink为RDS MySQL创建的数据库名
  'table-name' = 'orders',
  'username' = 'MySQLUsername',
  'password' = 'MySQLPassword'
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from jdbcSource;
```

5. 按照如下方式查看taskmanager.out文件中的数据结果：

   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

```
+I(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24
10:02:03,0001,Alice,330106)
+I(202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25
12:03:00,0002,Bob,330110)
```

常见问题

无

 版权所有 © 华为技术有限公司

3.3.1.5 Kafka 源表

功能描述

创建source流从Kafka获取数据，作为作业的输入数据。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

前提条件

- 确保已创建Kafka集群。
- 该场景作业需要运行在DLI的独享队列上，因此要与kafka集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 建表时数据类型的使用请参考**Format**章节。

语法格式

```
create table kafkaSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)
)
with (
  'connector' = 'kafka',
  'topic' = '',
  'properties.bootstrap.servers' = '',
  'properties.group.id' = '',
  'scan.startup.mode' = '',
  'format' = ''
);
```

参数说明

表 3-8 参数说明

参数	是否必选	默认值	数据类型	参数说明
connector	是	无	String	指定要使用的连接器，固定为'kafka'。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
topic	是	无	String	Kafka的topic名称。 注意： • “topic”和“topic-pattern”参数只能配置一个，不能同时配置。 • 若有多个topic，请以';'分隔，如'topic-1;topic-2'。
topic-pattern	否	无	String	匹配读取kafka topic名称的正则表达式。 注意：“topic-pattern”和“topic”只能选择一个，不可同时存在。 例如： 'topic.*' '(topic-c|topic-d)' '(topic-a|topic-b|topic-\\d*)' '(topic-a|topic-b|topic-[0-9]*)'
properties.bootstrap.servers	是	无	String	Kafka brokers地址，以逗号分隔。
properties.group.id	是	无	String	消费组名称
properties.*	否	无	String	设置和传入任意的Kafka原生配置文件。 注意： • “properties.”中的后缀名必须是Apache Kafka中的配置键。例如关闭自动创建topic：'properties.**allow.auto.create.topics**' = 'false'。 • 存在一些配置不支持配置，如'key.deserializer'和'value.deserializer'。
format	是	无	String	序列化和反序列化Kafka消息的value的格式。**注意：**该参数和'value.format'参数只能选择一个。 请参考**Format**页面以获取更多详细信息和格式参数。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
key.format	否	无	String	序列化和反序列化Kafka消息的key的格式。 注意： • 若配置了该参数，则'key.fields'也需要配置，否则kafka的记录中key会为空。 • 请参考**Format**页面以获取更多详细信息和格式参数。
key.fields	否	[]	List<String>	定义表中的列作为key的列表，同时需要配置'key.format'。 该参数默认为空，因此没有定义key。 使用形式如：'field1;field2'。
key.fields-prefix	否	无	String	为所有Kafka消息键（Key）指定自定义前缀，以避免与消息体（Value）格式字段重名。
value.format	是	无	String	用于反序列化和序列化 Kafka 消息的值部分的格式。 注意: • value.format和format参数只能配置其中一个，如果同时配置两个，则会有冲突。 • 请参考**Format**页面以获取更多详细信息和格式参数。
value.fields-include	否	ALL	枚举类型 可选值：[ALL, EXCEPT_KEY]	在解析消息体时，是否要包含消息键字段。 取值如下： • ALL（默认值）：所有定义的字段都存放消息体（Value）解析出来的数据。 • EXCEPT_KEY：除去key.fields定义字段，剩余的定义字段可以用来存放消息体（Value）解析出来的数据。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
scan.startup.mode	否	group-offsets	String	Kafka读取数据的启动位点。 取值如下： ● earliest-offset：从Kafka最早分区开始读取。 ● latest-offset：从Kafka最新位点开始读取。 ● group-offsets（默认值）：根据Group读取。 ● timestamp：从Kafka指定时间点读取。配置该参数时，同时需要在WITH参数中指定scan.startup.timestamp-millis参数。 ● specific-offsets：从Kafka指定分区指定偏移量读取。配置该参数时，同时需要在WITH参数中指定scan.startup.specific-offsets参数。
scan.startup.specific-offsets	否	无	String	在scan.startup.mode参数指定为'specific-offsets'模式下生效，为每个分区指定偏移量，例如：partition:0,offset:42;partition:1,offset:300'。
scan.startup.timestamp-millis	否	无	Long	在scan.startup.mode参数指定为'timestamp'模式下生效，指定启动位点时间戳。
scan.topic-partition-discovery.interval	否	无	Duration	消费者定期发现动态创建的Kafka主题和分区的时间间隔。
ssl_auth_name	否	无	String	DLI侧创建的Kafka_SSL类型的跨源认证名称。Kafka配置SSL时使用该配置。 注意：若仅使用SSL类型，则需要同时配置'properties.security.protocol '= 'SSL'; 若使用SASL_SSL类型，则需要同时配置 ● 'properties.security.protocol' = 'SASL_SSL'; ● 'properties.sasl.mechanism' = 'GSSAPI或者PLAIN'; ● 'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required username=\"xxx\" password=\"xxx\";'

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	参数说明
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。Kafka配置SASL认证时使用该配置。 注意：如果使用SASL_PLAINTEXT类型，且使用Kerberos认证，则需要同时配置'properties.sasl.mechanism' = 'GSSAPI'和'properties.security.protocol' = 'SASL_PLAINTEXT'

元信息列

您可以在源表中定义元信息列，以获取Kafka消息的元信息。例如，当WITH参数中定义了多个topic时，如果在Kafka源表中定义了元信息列，那么Flink读取到的数据就会被标识是从哪个topic中读取的数据。

表 3-9 元信息列

Key	数据类型	是否可读(R)写(W)	说明
topic	STRING NOT NULL	R	Kafka消息所在的Topic名称。
partition	INT NOT NULL	R	Kafka消息所在的Partition名称。
headers	MAP<STRING, BYTES> NOT NULL	R/W	Kafka消息的headers。
leader-epoch	INT NULL	R	Kafka消息的Leader epoch。 **其书写方式请参考示例1**。
offset	BIGINT NOT NULL	R	Kafka消息的偏移量（offset）。
timestamp	TIMESTAMP(3) WITH LOCAL TIME ZONE NOT NULL	R/W	Kafka消息的时间戳。

版权所有 © 华为技术有限公司

Key	数据类型	是否可读(R)写(W)	说明
timestamp-type	STRING NOT NULL	R	Kafka消息的时间戳类型： • NoTimestampType：消息中没有定义时间戳。 • CreateTime：消息产生的时间。 • LogAppendTime：消息被添加到Kafka Broker的时间。**其书写方式请参考示例1**。

示例（适用于 Kafka 集群未开启 SASL_SSL 场景）

- **示例1：读取Kafka的元信息列，输出到Print sink中。**
  a. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
  b. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
  c. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

  注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  `topic` String metadata,
  `partition` int metadata,
  `headers` MAP<STRING, BYTES> metadata,
  `leaderEpoch` INT metadata from 'leader-epoch',
  `offset` bigint metadata,
  `timestamp` TIMESTAMP(3) metadata,
  `timestampType` string metadata from 'timestamp-type',
  `message` string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  "format" = "csv",
  "csv.field-delimiter" = "\u0001",
  "csv.quote-character" = """
);

CREATE TABLE printSink (
  `topic` String,
  `partition` int,
  `headers` MAP<STRING, BYTES>,
  `leaderEpoch` INT,
  `offset` bigint,
```


版权所有 © 华为技术有限公司

```
  `timestamp` TIMESTAMP(3),
  `timestampType` string,
  `message` string --message表示读取kafka中存储的用户写入数据
) WITH (
  'connector' = 'print'
);

insert into printSink select * from orders;
```

若不需要读取整个message的消息，而是需要读取每个字段的值，则需要将使用如下语句：

```
CREATE TABLE orders (
  `topic` String metadata,
  `partition` int metadata,
  `headers` MAP<STRING, BYTES> metadata,
  `leaderEpoch` INT metadata from 'leader-epoch',
  `offset` bigint metadata,
  `timestamp` TIMESTAMP(3) metadata,
  `timestampType` string metadata from 'timestamp-type',
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE printSink (
  `topic` String,
  `partition` int,
  `headers` MAP<STRING, BYTES>,
  `leaderEpoch` INT,
  `offset` bigint,
  `timestamp` TIMESTAMP(3),
  `timestampType` string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from orders;
```

d. 向Kafka的相应topic中发送如下数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
```

```
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

e. 用户可按下述操作查看输出结果：

i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

数据结果参考如下：

```
+I(fz-source-json,0,{},0,243,2021-12-27T09:23:32.253,CreateTime,
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24
10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03",
"user_id":"0001", "user_name":"Alice", "area_id":"330106"})
+I(fz-source-json,0,{},0,244,2021-12-27T09:23:39.655,CreateTime,
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24
16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06",
"user_id":"0001", "user_name":"Alice", "area_id":"330106"})
+I(fz-source-json,0,{},0,245,2021-12-27T09:23:48.405,CreateTime,
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"})
```

- **示例2：将Kafka作为源表，Print作为结果表，从Kafka中读取编码格式为json数据类型的数据，输出到日志文件中。**

a. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。

b. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。

c. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，并提交运行。

注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
```


版权所有 © 华为技术有限公司

```
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from orders;
```

d. 向Kafka的相应topic中发送输入测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

e. 用户可按下述操作查看输出结果：

i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

数据结果参考如下：

```
+I(202103241000000001,webShop,2021-03-24T10:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,330106)
+I(202103241606060001,appShop,2021-03-24T16:06:06,200.0,180.0,2021-03-2416:10:06,0001,Alice,330106)
+I(202103251202020001,miniAppShop,2021-03-25T12:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
```

示例（适用于 Kafka 集群已开启 SASL_SSL 场景）

- **示例1：DMS集群使用SASL_SSL认证方式。**

  创建DMS的kafka集群，开启SASL_SSL，并下载SSL证书，将下载的证书client.jks上传到OBS桶中。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks',  -- 用户上传证书的位置
  'properties.sasl.mechanism' = 'PLAIN',  -- 按照SASL_PLAINTEXT方式填写
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required username=\"xx\" password=\"xx\";',  -- 创建kafka集群时设置的账号和密码
```

```
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks',
  'properties.sasl.mechanism' = 'PLAIN',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required
username=\"xx\" password=\"xx\";',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **示例2：MRS集群使用kafka SASL_SSL认证方式。**
  - MRS集群请开启Kerberos认证。
  - 在”组件管理 > Kafka > 服务配置”中查找配置项” security.protocol”，并设置为”SASL_SSL”。
  - 登录MRS集群的Manager，下载用户凭据：”系统设置 > 用户管理 ，单击用户名后的”更多 > 下载认证凭据”。

    根据用户凭据生成相应的truststore.jks文件，并将用户凭据以及truststore.jks文件传入OBS中。

    具体方式请参考**客户端SSL加密功能使用说明**。
  - 若运行作业提示“Message stream modified (41)”，可能与JDK的版本有关系，可以尝试修改运行样例代码的JDK为8u_242以下版本或删除“krb5.conf”配置文件的“renew_lifetime = 0m”配置项。
  - 端口请使用KafKa服务配置中设置的sasl_ssl.port端口。
  - security.protocol请设置为SASL_SSL。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',  -- 用户名
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
```

```
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',  -- 生成truststore.jks设置的密码
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **示例3：MRS集群使用SASL_PAINTEXT的Kerberos认证。**
  - MRS集群请开启Kerberos认证。
  - 将“组件管理 > Kafka > 服务配置”中查找配置项” security.protocol”，并设置为”SASL_PLAINTEXT”。
  - 登录MRS集群的Manager，下载用户凭据“系统设置 > 用户管理”，单击用户名后的“更多 > 下载认证凭据”，并上传到OBS中。
  - 若运行提示“Message stream modified (41)”的错误，可能与JDK的版本有关系，可以尝试修改运行样例代码的JDK为8u_242以下版本或删除“krb5.conf”配置文件的“renew_lifetime = 0m”配置项。
  - 端口请使用KafKa服务配置中设置的sasl.port端口。
  - security.protocol请设置为SASL_PLAINTEXT。

```
CREATE TABLE ordersSources (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21007,xx:21007',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.sasl.kerberos.service.name' = 'kafka',
```

```
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21007,xx:21007',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **示例4：MRS集群使用SSL方式。**
  - MRS集群请不要开启Kerberos认证。
  - 登录MRS集群的Manager，下载用户凭据：“系统设置 > 用户管理”。单击用户名后的“更多 > 下载认证凭据”。

    根据用户凭据生成相应的truststore.jks文件，并将用户凭据以及truststore.jks文件传入OBS中。

    具体方式请参考**客户端SSL加密功能使用说明**。
  - 端口请注意使用KafKa服务配置中设置的ssl.port端口
  - security.protocol请设置为SSL。
  - ssl.mode.enable请设置为true。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',  -- 生成truststore.jks时设置的密码
  'properties.security.protocol' = 'SSL',
```

```
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into ordersSink select * from ordersSource;
```

常见问题

- **Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?**

  ```
  org.apache.kafka.common.errors.TimeoutException: Timeout expired while fetching topic metadata
  ```

  跨源未绑定或未绑定成功，或是Kafka集群安全组未配置放通DLI队列的网段地址。参考**增强型跨源连接**重新配置跨源，或者Kafka集群安全组放通DLI队列的网段地址。

- **Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?**

  ```
  Caused by: java.lang.RuntimeException: RealLine:45;Table 'default_catalog.default_database.printSink' declares persistable metadata columns, but the underlying DynamicTableSink doesn't implement the SupportsWritingMetadata interface. If the column should not be persisted, it can be declared with the VIRTUAL keyword.
  ```

  sink表中定义了metadata类型，但是Print connector并不支持把sink表中的matadata去掉即可。

3.3.1.6 MySQL CDC 源表

功能描述

MySQL的CDC源表，即MySQL的流式源表，会先读取数据库的历史全量数据，并平滑切换到Binlog读取上，保证数据的完整读取。

前提条件

- MySQL CDC要求MySQL版本为5.7或8.0.x。
- 该场景作业需要DLI与MySQL建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。
- MySQL已开启了Binlog，并且binlog_row_image设置为FULL。
- 已创建MySQL用户，并授予了SELECT、 SHOW DATABASES 、REPLICATION SLAVE和REPLICATION CLIENT权限。在 scan.incremental.snapshot.enabled 参数已启用时（默认情况下已启用）时，还需要授予RELOAD权限。

 版权所有 © 华为技术有限公司

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 同步数据库数据的客户端，都会有一个唯一ID，即Server ID。同一个数据库下，建议每个MySQL CDC作业配置不同的Server ID。

  主要原因如下：

  - MySQL SERVER会根据该ID来维护网络连接以及Binlog位点。因此如果有大量相同的Server ID的客户端一起连接MySQL SERVER，可能导致MySQL SERVER的CPU陡增，影响线上业务稳定性。
  - 此外，多个作业共享相同的Server ID，会导致Binlog位点错乱，多读或少读数据，因此建议每个CDC作业都配置不同的Server ID。
- MySQL CDC源表暂不支持定义Watermark。如果您需要进行窗口聚合，请参考**常见问题**描述。
- 若连接DWS、MySQL等支持upsert的sink源，需要在sink表的创建语句中定义主键，请参考**示例**中printSink建表语句。
- 当使用MySQ CDM源表时，请不要在源表参数里手动关闭debezium.connect.keep.alive，确保debezium.connect.keep.alive=true（默认值为true）。

  如果手动关闭了debezium.connect.keep.alive，一旦发生拉取Binlog线程与MySQL服务器的连接连接异常，拉取Binlog线程不会尝试自动重连，这可能导致无法正常从源端拉取binlog日志。

语法格式

```
create table mySqlCdcSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'mysql-cdc',
  'hostname' = 'mysqlHostname',
  'username' = 'mysqlUsername',
  'password' = 'mysqlPassword',
  'database-name' = 'mysqlDatabaseName',
  'table-name' = 'mysqlTableName'
);
```

参数说明

表 3-10 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'mysql-cdc'。
hostname	是	无	String	MySQL数据库的IP地址或者Hostname。
username	是	无	String	MySQL数据库的用户名。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
password	是	无	String	MySQL数据库的密码。
database-name	是	无	String	访问的数据库名称。 数据库名称支持正则表达式以读取多个数据库的数据，例如flink(.)*表示以flink开头的数据库名。
table-name	是	无	String	访问的表名。 表名支持正则表达式以读取多个表的数据，例如cdc_order(.)*表示以cdc_order开头的表名。
port	否	3306	Integer	MySQL数据库的端口号。
server-id	否	5400~6000随机值	String	数据库客户端的一个数字ID，该ID必须是MySQL集群中全局唯一的。建议针对同一个数据库的每个作业都设置一个不同的ID。 默认会随机生成一个5400~6400的值。
scan.startup.mode	否	initial	String	消费数据时的启动模式。 • initial（默认）：在第一次启动时，会先扫描历史全量数据，然后读取最新的Binlog数据。 • latest-offset：在第一次启动时，不会扫描历史全量数据，直接从Binlog的末尾（最新的Binlog处）开始读取，即只读取该Connector启动以后的最新变更。
server-time-zone	否	无	String	数据库在使用的会话时区。 例如：Asia/Shanghai。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置置账号和密码。

示例

该示例是利用MySQL-CDC实时读取RDS MySQL中的数据，并写入到Print结果表中，其具体步骤如下（本示例使用RDS MySQL数据库引擎版本为MySQL 5.7.32）。

1. 参考**增强型跨源连接**，根据MySQL所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。

版权所有 © 华为技术有限公司

2. 设置MySQL的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据MySQL的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 在MySQL中的flink数据库下创建相应的表，表名为cdc_order，SQL语句参考如下：

```
CREATE TABLE `flink`.`cdc_order` (
    `order_id` VARCHAR(32) NOT NULL,
    `order_channel` VARCHAR(32) NULL,
    `order_time` VARCHAR(32) NULL,
    `pay_amount` DOUBLE  NULL,
    `real_pay` DOUBLE  NULL,
    `pay_time` VARCHAR(32) NULL,
    `user_id` VARCHAR(32) NULL,
    `user_name` VARCHAR(32) NULL,
    `area_id` VARCHAR(32) NULL,
    PRIMARY KEY (`order_id`)
)   ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4
    COLLATE = utf8mb4_general_ci;
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
create table mysqlCdcSource(
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id STRING
) with (
  'connector' = 'mysql-cdc',
  'hostname' = '***mysqlHostname***',
  'username' = '***mysqlUsername***',
  'password' = '***mysqlPassword***',
  'database-name' = '***mysqlDatabaseName***',
  'table-name' = '***mysqlTableName***'
);

create table printSink(
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id STRING,
  primary key(order_id) not enforced
) with (
  'connector' = 'print'
);

insert into printSink select * from mysqlCdcSource;
```

5. 在MySQL中执行以下命令插入测试数据。

```
insert into cdc_order values
('202103241000000001','webShop','2021-03-24 10:00:00','100.00','100.00','2021-03-24
10:02:03','0001','Alice','330106'),
('202103241606060001','appShop','2021-03-24 16:06:06','200.00','180.00','2021-03-24
16:10:06','0001','Alice','330106');
```


版权所有 © 华为技术有限公司

```
delete from cdc_order  where order_channel = 'webShop';

insert into cdc_order values('202103251202020001','miniAppShop','2021-03-25
12:02:02','60.00','60.00','2021-03-25 12:03:00','0002','Bob','330110');
```

6. 按照如下方式查看taskmanager.out文件中的数据结果：

   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

```
+I(202103241000000001,webShop,2021-03-2410:00:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,330
106)
+I(202103241606060001,appShop,2021-03-2416:06:06,200.0,180.0,2021-03-2416:10:06,0001,Alice,3301
06)
-
D(202103241000000001,webShop,2021-03-2410:00:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,330
106)
+I(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330
110)
```

常见问题

Q：MySQL CDC源表不支持定义Watermark，怎么进行窗口聚合?

A：可以采用非窗口聚合的方式，即将时间字段转换成窗口值，然后根据窗口值进行GROUP BY聚合。

例如：基于上述示例，统计每分钟的订单数，脚本如下（其中order_time为string类型，表示订单的时间）。

```
insert into printSink select DATE_FORMAT(order_time, 'yyyy-MM-dd HH:mm'), count(*) from
mysqlCdcSource group by DATE_FORMAT(order_time, 'yyyy-MM-dd HH:mm');
```

3.3.1.7 Postgres CDC 源表

功能描述

Postgres的CDC源表，即Postgres的流式源表，用于依次读取PostgreSQL数据库全量快照数据和变更数据，保证不多读一条也不少读一条数据。即使发生故障，也能采用Exactly Once方式处理。

前提条件

- PostgreSQL的版本不能低于PostgreSQL 11。
- 要与实例建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

版权所有 © 华为技术有限公司

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 若Postgres表有update等操作，需要在PostgreSQL中执行下列语句。注意：test.cdc_order需要修改为实际的数据库和表。
  ```
  ALTER TABLE test.cdc_order REPLICA IDENTITY FULL
  ```
- 使用前请确认当前PostgreSQL是否包含默认的插件，可在PostgreSQL中使用下述语句查询当前插件。
  ```
  SELECT name FROM pg_available_extensions;
  ```
  若不包含默认插件名“decoderbufs”，则需要在创建PostgreSQL CDC源表中配置参数“decoding.plugin.name”，该参数指定PostgreSQL中已有的插件。

语法格式

```
create table postgresCdcSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'postgres-cdc',
  'hostname' = 'PostgresHostname',
  'username' = 'PostgresUsername',
  'password' = 'PostgresPassword',
  'database-name' = 'PostgresDatabaseName',
  'schema-name' = 'PostgresSchemaName',
  'table-name' = 'PostgresTableName'
);
```

参数说明

表 3-11 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'postgres-cdc'。
hostname	是	无	String	Postgres数据库的IP地址或者Hostname。
username	是	无	String	Postgres数据库用户名。
password	是	无	String	Postgres数据库服务的密码。
database-name	是	无	String	数据库名称。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
schema-name	是	无	String	Postgres Schema名称。 Schema名称支持正则表达式以读取多个Schema的数据，例如test(.)*表示以test开头的所有schema。
table-name	是	无	String	Postgres表名。 表名支持正则表达式去读取多个表的数据，例如cdc_order(.)*表示以cdc_order开头的所有表。
port	否	5432	Integer	Postgres数据库服务的端口号。
decoding.plugin.name	否	decoderbufs	String	根据Postgres服务上安装的插件确定。支持的插件列表如下： • decoderbufs（默认值） • wal2json • wal2json_rds • wal2json_streaming • wal2json_rds_streaming • pgoutput
debezium.*	否	无	String	更细粒度控制Debezium客户端的行为。例如'debezium.snapshot.mode' = 'never'。 建议每个表都设置debezium.slot.name参数，以避免出现 “PSQLException: ERROR: replication slot "debezium" is active for PID 974”报错。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置账号和密码。

示例

该示例是利用Postgres-CDC实时读取RDS PostgreSQL中的数据，并写入到Print结果表中，其具体步骤如下（当前示例使用的数据库引擎版本是RDS PostgreSQL 11.11）：

1. 参考**增强型跨源连接**，根据PostgreSQL所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置PostgreSQL的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据PostgreSQL的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。

版权所有 © 华为技术有限公司

3. 在PostgreSQL中创建数据库flink，并创建名为test的schema。
4. 在PostgreSQL中flink数据库的test schema下创建表名为cdc_order的表，SQL语句参考如下：

```
create table test.cdc_order(
  order_id VARCHAR,
  order_channel VARCHAR,
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR,
  primary key(order_id)
);
```

5. 在PostgreSQL中执行下列SQL语句。如果不执行如下命令，后续Flink作业将会运行报错，具体报错信息详情参见**错误信息**。

```
ALTER TABLE test.cdc_order REPLICA IDENTITY FULL
```

6. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
create table postgresCdcSource(
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id STRING,
  primary key (order_id) not enforced
) with (
  'connector' = 'postgres-cdc',
  'hostname' = '***PostgresHostname***',
  'username' = '***PostgresUsername***',
  'password' = '***PostgresPassword***',
  'database-name' = 'flink',
  'schema-name' = 'test',
  'table-name' = 'cdc_order'
);

create table printSink(
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id STRING,
  primary key(order_id) not enforced
) with (
  'connector' = 'print'
);

insert into printSink select * from postgresCdcSource;
```

7. 在PostgreSQL中执行以下命令：

```
insert into test.cdc_order
  (order_id,
  order_channel,
```

 版权所有 © 华为技术有限公司

```
  order_time,
  pay_amount,
  real_pay,
  pay_time,
  user_id,
  user_name,
  area_id) values
  ('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',
'0001', 'Alice', '330106'),
  ('202103251202020001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',
'0002', 'Bob', '330110');

update test.cdc_order set order_channel = 'webShop' where order_id = '202103251202020001';

delete from test.cdc_order where order_id = '202103241000000001';
```

8. 按照如下方式查看taskmanager.out文件中的数据结果：

   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

```
+I(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24
10:02:03,0001,Alice,330106)
+I(202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25
12:03:00,0002,Bob,330110)
-U(202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25
12:03:00,0002,Bob,330110)
+U(202103251202020001,webShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25
12:03:00,0002,Bob,330110)
-D(202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24
10:02:03,0001,Alice,330106)
```

常见问题

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

```
org.postgresql.util.PSQLException: ERROR: logical decoding requires wal_level >= logical
```

- A：需要调节PostgreSQL的配置参数wal_level为logical，并重新启动。具体操作可以参考**设置wal_level参数为logical**。

  PostgreSQL参数修改完成后，需要重启下RDS PostgreSQL实例，使得参数生效。

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

```
java.lang.IllegalStateException: The "before" field of UPDATE/DELETE message is null, please check
the Postgres table has been set REPLICA IDENTITY to FULL level. You can update the setting by
running the command in Postgres 'ALTER TABLE test.cdc_order REPLICA IDENTITY FULL'.
```

  A：若运行日志出现类似报错问题，则需要在PostgreSQL中执行报错日志中的语句"ALTER TABLE test.cdc_order REPLICA IDENTITY FULL"。

3.3.1.8 Redis 源表

功能描述

创建source流从Redis获取数据，作为作业的输入数据。

前提条件

- 创建该作业前，需要建立DLI和Redis的增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 若需要获取key的值，则可以通过在Flink中设置主键获取，主键字段即对应Redis的key。
- 若定义主键，则不能够定义复合主键，即主键只能是一个字段，不能是多个字段。
- schema-syntax取值约束：
  - 当schema-syntax为map或array时，非主键字段最多只能有一个，且需要为相应的map或array类型。
  - 当schema-syntax为fields-scores时，非主键字段个数需要为偶数，且除主键字段外，每两个字段的第二个字段的数据类型需要为doule，该字段的值视为前一个字段的score。其示例如下：

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  score1 double,
  order_channel string,
  score2 double,
  order_time string,
  score3 double,
  pay_amount double,
  score4 double,
  real_pay double,
  score5 double,
  pay_time string,
  score6 double,
  user_id string,
  score7 double,
  user_name string,
  score8 double,
  area_id string,
  score9 double,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields-scores'
);
```

- data-type取值约束：
  - 当data-type为set时，Flink中定义的非主键字段的数据类型必须相同。
  - 当data-type为sorted-set并且schema-syntax为fields和array时，只能读取redis的sorted set中的值，而不能读取score。
  - 当data-type为string时，只能有一个非主键字段。

版权所有 © 华为技术有限公司

- 当data-type为sorted-set，且schema-syntax为map时，除主键字段外，只能有一个非主键字段。

  该非主键字段需要为map类型，同时该字段map的value需要为double类型，表示score，该字段的map的key表示redis的set中的值。

- 当data-type为sorted-set，且schema-syntax为array-scores时，除主键字段外，只能有两个非主键字段，且这两个字段的类型需要为array。

  两个字段其中第一个字段类型是array，表示Redis的set中的值；第二个字段类型为array<double>，表示相应索引的score。其示例如下：

```
CREATE TABLE redisSink (
  order_id string,
  arrayField Array<String>,
  arrayScore array<double>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
  "default-score" = '3',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'array-scores'
);
```

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
  ,PRIMARY KEY (attr_name, ...) NOT ENFORCED
)
with (
  'connector' = 'redis',
  'host' = ''
);
```

参数说明

表 3-12 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'redis'。
host	是	无	String	redis连接地址。
port	否	6379	Integer	redis连接端口。
password	否	无	String	redis认证密码。
namespace	否	无	String	redis key的namespace
delimiter	否	:	String	redis的key和namespace之间的分隔符。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
data-type	否	hash	String	redis的数据类型，有下列选项： • hash • list • set • sorted-set • string data-type取值约束详见**data-type取值约束**说明。
schema-syntax	否	fields	String	redis的schema语义，包含以下值（其具体使用请参考**注意事项**和**常见问题**）： • fields：适用于所有数据类型 • fields-scores：适用于sorted set数据类型 • array：适用于list、set、sorted set数据类型 • array-scores：适用于sorted set数据类型 • map：适用于hash、sorted set数据类型 schema-syntax取值约束详见**schema-syntax取值约束**说明。
deploy-mode	否	standalone	String	redis集群的部署模式，支持standalone、master-replica、cluster。默认为standalone。
retry-count	否	5	Integer	连接redis集群的尝试次数。
connection-timeout-millis	否	10000	Integer	尝试连接redis集群时的最大超时时间。
commands-timeout-millis	否	2000	Integer	等待操作完成响应的最大时间。
rebalancing-timeout-millis	否	15000	Integer	redis集群失败时的休眠时间。
scan-keys-count	否	1000	Integer	每次扫描时读取的数量。
default-score	否	0	Double	当data-type设置为“sorted-set”时的默认score。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
deserialize-error-policy	否	fail-job	Enum	数据解析失败时的处理方式。枚举类型，包含以下值： • fail-job：作业失败 • skip-row：跳过当前数据 • null-field：设置当前数据为null
skip-null-values	否	true	Boolean	是否跳过null。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置账号和密码。

示例

该示例是从DCS Redis数据源中读取数据，并写入Print到结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，根据redis所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Redis的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据redis的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 在Redis客户端中执行如下命令，向不同的key中插入数据，以hash形式存储：

```
HMSET redisSource order_id 202103241000000001 order_channel webShop order_time "2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24 10:02:03" user_id 0001 user_name Alice area_id 330106

HMSET redisSource1 order_id 202103241606060001 order_channel appShop order_time "2021-03-24 16:06:06" pay_amount 200.00 real_pay 180.00 pay_time "2021-03-24 16:10:06" user_id 0001 user_name Alice area_id 330106

HMSET redisSource2 order_id 202103251202020001 order_channel miniAppShop order_time "2021-03-25 12:02:02" pay_amount 60.00 real_pay 60.00 pay_time "2021-03-25 12:03:00" user_id 0002 user_name Bob area_id 330110
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本读取Redis中hash格式的数据。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
```


版权所有 © 华为技术有限公司

```
  primary key (redisKey) not enforced  --获取redis中key的值
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'hash',
  'deploy-mode' = 'master-replica'
);

CREATE TABLE printSink (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from redisSource;
```

5. 按照如下方式查看taskmanager.out文件中的数据结果：

   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。

   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

   c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   数据结果参考如下：

```
+I(redisSource1,202103241606060001,appShop,2021-03-24 16:06:06,200.0,180.0,2021-03-24 16:10:06,0001,Alice,330106)
+I(redisSource,202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
+I(redisSource2,202103251202020001,miniAppShop,2021-03-25 12:02:02,60.0,60.0,2021-03-25 12:03:00,0002,Bob,330110)
```

常见问题

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

  ```
  Caused by: org.apache.flink.client.program.ProgramInvocationException: The main method caused an error: RealLine:36;Usage of 'set' data-type and 'fields' schema syntax in source Redis connector with multiple non-key column types. As 'set' in Redis is not sorted, it's not possible to map 'set's values to table schema with different types.
  ```

  A：data-type为set类型时，flink中非主键字段的数据类型不相同，导致如上报错。data-type为set类型时，Flink中定义的非主键字段的数据类型必须相同。

- Q：当使用data-type为hash时，那么schema-syntax为fields和map有什么区别?

  A：当schema-syntax为fields时，会将Redis的key中hash值赋给flink中同名相应字段；当schema-syntax为map时，会将Redis的每个hash中的hashkey和hashvalue放入一个map中，该map即为flink中相应字段的值，即这个map中包含Redis中某个key的所有hashkey和hashvalue。

  – 对于fields而言：

    i. 向Redis中插入如下数据

    ```
    HMSET redisSource order_id 202103241000000001 order_channel webShop order_time "2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24 10:02:03" user_id 0001 user_name Alice area_id 330106
    ```

 版权所有 © 华为技术有限公司

ii. 当使用schema-syntax为fields时，作业脚本参考如下：

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'hash',
  'deploy-mode' = 'master-replica'
);

CREATE TABLE printSink (
  redisKey string,
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from redisSource;
```

iii. 作业运行结果如下：

```
+I(redisSource,202103241000000001,webShop,2021-03-24
10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106)
```

- 对于map而言：

i. 向Redis中插入如下数据：

```
HMSET redisSource order_id 202103241000000001 order_channel webShop order_time
"2021-03-24 10:00:00" pay_amount 100.00 real_pay 100.00 pay_time "2021-03-24
10:02:03" user_id 0001 user_name Alice area_id 330106
```

ii. 当使用schema-syntax为map时，其作业脚本参考如下：

```
CREATE TABLE redisSource (
  redisKey string,
  order_result map<string, string>,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'hash',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'map'
);

CREATE TABLE printSink (
  redisKey string,
  order_result map<string, string>
) WITH (
  'connector' = 'print'
);

insert into printSink select * from redisSource;
```

 版权所有 © 华为技术有限公司

iii. 作业运行结果如下：

```
+I(redisSource,{user_id=0001, user_name=Alice, pay_amount=100.00, real_pay=100.00, order_time=2021-03-24 10:00:00, area_id=330106, order_id=202103241000000001, order_channel=webShop, pay_time=2021-03-24 10:02:03})
```

3.3.1.9 Upsert Kafka 源表

功能描述

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

作为 source，upsert-kafka 连接器生产changelog流，其中每条数据记录代表一个更新或删除事件。更准确地说，数据记录中的 value 被解释为同一 key 的最后一个 value 的 UPDATE，如果有这个 key（如果不存在相应的 key，则该更新被视为 INSERT）。用表来类比，changelog 流中的数据记录被解释为 UPSERT，也称为 INSERT/UPDATE，因为任何具有相同 key 的现有行都被覆盖。另外，value 为空的消息将会被视作为 DELETE 消息。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与kafka集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- Upsert Kafka 始终以upsert方式工作，并且需要在DDL中定义主键。在具有相同主键值的消息按序存储在同一个分区的前提下，在 changlog source 定义主键意味着 在物化后的 changelog 上主键具有唯一性。定义的主键将决定哪些字段出现在Kafka消息的key中。
- 由于该连接器以 upsert 的模式工作，该连接器作为 source 读入时，可以确保具有相同主键值下仅最后一条消息会生效。
- 数据类型的使用，请参考**Format**章节。

语法格式

```
create table kafkaSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'upsert-kafka',
  'topic' = '',
```

 版权所有 © 华为技术有限公司

```
  'properties.bootstrap.servers' = '',
  'key.format' = '',
  'value.format' = ''
);
```

参数说明

表 3-13 参数说明

参数	是否必选	默认参数	数据类型	说明
connector	是	无	String	connector类型，对于upsert kafka，需配置为'upsert-kafka'。
topic	是	无	String	Kafka topic名。
properties.bootstrap.servers	是	无	String	Kafka brokers地址，以逗号分隔。
key.format	是	无	String	用于对Kafka消息中key部分序列化和反序列化的格式。key字段由PRIMARY KEY语法指定。支持的格式如下： ● csv ● json ● avro 请参考**Format**页面以获取更多详细信息和格式参数。
key.fields-prefix	否	无	String	为键格式的所有字段定义自定义前缀，以避免与值格式的字段发生名称冲突。 默认情况下，前缀为空。如果定义了自定义前缀，则表架构和'key.fields'都将使用前缀名称。在构造密钥格式的数据类型时，将删除前缀，并在密钥格式中使用无前缀的名称。请注意，此选项要求'value.fields-include' 必须设置为'EXCEPT_KEY'。
value.format	是	无	String	用于对 Kafka消息中 value 部分序列化和反序列化的格式。支持的格式： ● csv ● json ● avro 请参考**Format**页面以获取更多详细信息和格式参数。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
value.fields-include	是	ALL	String	控制哪些字段应该出现在值中。取值范围如下： • ALL：消息的value部分将包含schema的所有字段，包括定义中键的字段。 • EXCEPT_KEY：记录的value部分包含schema的所有内容，定义为主键的字段除外。
properties.*	否	无	String	该选项可以传递任意的Kafka参数。 “properties.”后的后缀名必须匹配定义在 **kafka参数文档**中的参数名。 Flink会自动移除选项名中的 "properties." 前缀，并将转换后的键名以及值传入KafkaClient。 例如：您可以通过 'properties.allow.auto.create.topics' = 'false' 来禁止自动创建 topic。 但是'key.deserializer' 和 'value.deserializer' 是不允许通过该方式传递参数，因为Flink会重写这些参数的值。
ssl_auth_name	否	无	String	DLI侧创建的Kafka_SSL类型的跨源认证名称。Kafka配置SSL时使用该配置。 注意：若仅使用SSL类型，则需要同时配置'properties.security.protocol '= 'SSL'; 若使用SASL_SSL类型，则需要同时配置 'properties.security.protocol' = 'SASL_SSL'、 'properties.sasl.mechanism' = 'GSSAPI或者PLAIN'、 'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required username=\"xxx\" password=\"xxx\";'
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。Kafka配置SASL认证时使用该配置。 注意：如果使用SASL_PLAINTEXT类型，且使用Kerberos认证，则需要同时配置 'properties.sasl.mechanism' = 'GSSAPI' 和'properties.security.protocol' = 'SASL_PLAINTEXT'

示例

该示例是从Kafka数据源中读取数据，并写入Print到结果表中，其具体步骤如下：

版权所有 © 华为技术有限公司

1. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE upsertKafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' =  '***KafkaAddress1:KafkaPort,KafkAddress2:KafkaPort***',
  'key.format' = 'csv',
  'value.format' = 'json'
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'print'
);

INSERT INTO printSink
SELECT * FROM upsertKafkaSource;
```

4. 向Kafka中的指定topic中插入如下数据（注意：**kafka插入数据时请指定key**）。

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}


{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

5. 用户可按下述操作查看输出结果：
   a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
   b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。

 版权所有 © 华为技术有限公司

c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

数据结果参考如下：

```
+I(20210325120202001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
+I(20210325150505001,qqShop,2021-03-2515:05:05,500.0,400.0,2021-03-2515:10:00,0003,Cindy,330108)
-U(20210325120202001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
+U(20210325120202001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
```

常见问题

无

3.3.1.10 FileSystem 源表

功能描述

本节介绍FileSystem源表的定义，以及创建源表时使用的参数和示例代码。

前提条件

该场景作业需要建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 数据类型的使用，请参考**Format**章节。

语法格式

```
create table dataGenSource(
  amount int
) with (
  'connector' = 'filesystem',
  'path' = 'obs://longyuan/source-files',
  'format' = 'csv'
);
```


版权所有 © 华为技术有限公司

参数说明

表 3-14 参数说明

参数	是否必选	默认参数	数据类型	说明
connector	是	无	String	固定位filesystem。
path	是	无	String	OBS路径。
format	是	无	String	文件格式。 支持csv、parquet格式。

常见问题

无

3.3.2 创建结果表

3.3.2.1 BlackHole 结果表

功能描述

BlackHole Connector允许接收所有输入记录，常用于高性能测试和UDF 输出，其不是实质性Sink。Blackhole结果表是系统内置的Connector。

例如，如果您在注册其他类型的Connector结果表时报错，但您不确定是系统问题还是结果表WITH参数错误，您可以将WITH参数修改为'connector' = 'blackhole'后，单击运行。如果不再报错，则证明系统没有问题，您需要排查确认修改WITH参数是否正确。

前提条件

无

注意事项

创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table blackhole_table (
 attr_name attr_type (',' attr_name attr_type) *
) with (
 'connector' = 'blackhole'
);
```

 版权所有 © 华为技术有限公司

参数说明

表 3-15

选项	是否必要	默认值	类型	描述
connector	是	无	String	指定需要使用的连接器，此处应为'blackhole'。

示例

通过DataGen源表产生数据，BlackHole结果表接收传来的数据。

```
create table datagenSource (
 user_id string,
 user_name string,
 user_age int
) with (
 'connector' = 'datagen',
 'rows-per-second'='1'
);
create table blackholeSink (
 user_id string,
 user_name string,
 user_age int
) with (
 'connector' = 'blackhole'
);
insert into blackholeSink select * from datagenSource;
```

3.3.2.2 ClickHouse 结果表

功能描述

DLI支持将Flink作业数据输出到ClickHouse数据库中。ClickHouse是面向联机分析处理的列式数据库，支持SQL查询，且查询性能好，特别是基于大宽表的聚合分析查询性能非常优异，比其他分析型数据库速度快一个数量级。详细请参考**ClickHouse组件操作**。

前提条件

- 该场景需要与ClickHouse建立增强型跨源连接，并根据实际情况设置ClickHouse集群所在安全组规则中的端口。

  如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。

  如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 创建MRS的ClickHouse集群，集群版本选择MRS 3.1.0及以上版本，且勿开启kerberos认证。

版权所有 © 华为技术有限公司

- ClickHouse结果表不支持删除表数据操作。
- Flink中支持字段类型范围为：string、tinyint、smallint、int、long、float、double、date、timestamp、decimal以及Array。

  其中Array中的数据类型仅支持int、bigint、string、float、double。

语法格式

```
create table clickhouseSink (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = clickhouse,
  'connector.url' = '',
  'connector.table' = ''
);
```

参数说明

表 3-16 参数说明

参数	是否必选	默认值	数据类型	说明
connector.type	是	无	String	固定为：clickhouse
connector.url	是	无	String	ClickHouse的url。 参数格式为：**jdbc:clickhouse://**ClickHouseBalancer实例的IP**:**ClickHouseBalancer实例的http端口**/**数据库名 ● ClickHouseBalancer实例的IP地址： 登录MRS管理控制台，选择“集群名称 > 组件管理 > ClickHouse > 实例”，获取ClickHouseBalancer实例的业务IP。 ● ClickHouseBalancer实例的http端口： 登录MRS管理控制台，选择“集群名称 > 组件管理 > ClickHouse > 服务配置”，角色选择“ClickHouseBalancer”，搜索“lb_http_port”配置参数值。默认为：21425。 ● 数据库名为ClickHouse集群创建的数据库名称。
connector.table	是	无	String	要创建的ClickHouse的表名。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
connector.driver	否	ru.yandex.clickhouse.ClickHouseDriver	String	连接数据库所需要的驱动。 • 如果建表时不指定该参数，驱动会自动通过ClickHouse的url提取。 • 如果建表时指定该参数，则该参数值固定为“ru.yandex.clickhouse.ClickHouseDriver”。
connector.username	否	无	String	访问ClickHouse数据库的账号。
connector.password	否	无	String	访问ClickHouse数据库账号的密码。
connector.write.flush.max-rows	否	5000	Integer	写数据时刷新数据的最大行数，默认值为：5000。
connector.write.flush.interval	否	0	Duration	刷新数据的时间间隔，单位可以为ms、milli、millisecond/s、sec、second/min、minute等。 为0则表示不根据时间刷新
connector.write.max-retries	否	3	Integer	写数据失败时的最大尝试次数，默认值为：3。

示例

从Kafka中读取数据，并将数据插入到数据库为flink、表名为order的ClickHouse数据库中，其具体步骤如下（clickhouse版本为MRS的21.3.4.25）：

1. 参考**增强型跨源连接**，在DLI上根据ClickHouse和Kafka集群所在的虚拟私有云和子网分别创建跨源连接，并绑定所要使用的Flink作业队列。
2. 设置ClickHouse和Kafka集群安全组的入向规则，使其对当前将要使用的Flink作业队列网段放通。参考**测试地址连通性**根据ClickHouse和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 使用ClickHouse客户端连接到ClickHouse服务端，并使用以下命令查询集群标识符cluster等其他环境参数信息。

   ```
   select cluster,shard_num,replica_num,host_name from system.clusters;
   ```

   其返回信息如下图：

   ```
   ┌─cluster─────┬─shard_num─┐
   │ default_cluster │ 1 │ 1 │
   │ default_cluster │ 1 │ 2 │
   └─────────┴─────┴─────┘
   ```

4. 根据获取到的集群标识符cluster，例如当前为default_cluster ，使用以下命令在ClickHouse的default_cluster集群节点上创建数据库flink。

   ```
   CREATE DATABASE flink ON CLUSTER default_cluster;
   ```

 版权所有 © 华为技术有限公司

5. 使用以下命令在default_cluster集群节点上和flink数据库下创建表名为order的ReplicatedMergeTree表。

```
CREATE TABLE flink.order ON CLUSTER default_cluster(order_id String,order_channel String,order_time String,pay_amount Float64,real_pay Float64,pay_time String,user_id String,user_name String,area_id String) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/flink/order', '{replica}')ORDER BY order_id;
```

6. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将Kafka作为数据源，ClickHouse作业结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

create table clickhouseSink(
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) with (
  'connector.type' = 'clickhouse',
  'connector.url' = 'jdbc:clickhouse://***ClickhouseAddress:ClickhousePort***/flink',
  'connector.table' = 'order',
  'connector.write.flush.max-rows' = '1'
);

insert into clickhouseSink select * from orders;
```

7. 连接Kafka集群，向Kafka中插入以下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

8. 使用ClickHouse客户端连接到ClickHouse，执行以下查询命令，查询写入flink数据库下order表中的数据。

```
select * from flink.order;
```

   查询结果参考如下：

 版权所有 © 华为技术有限公司

```
202103241000000001 webShop 2021-03-24 10:00:00 100 100 2021-03-24 10:02:03 0001 Alice 330106

202103241606060001 appShop 2021-03-24 16:06:06 200 180 2021-03-24 16:10:06 0001 Alice 330106

202103251202020001 miniAppShop 2021-03-25 12:02:02 60 60 2021-03-25 12:03:00 0002 Bob 330110
```

常见问题

无

3.3.2.3 DWS 结果表

功能描述

DLI将Flink作业的输出数据输出到数据仓库服务（DWS）中。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见**《数据仓库服务管理指南》**。

前提条件

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。如何创建DWS集群，请参考《数据仓库服务管理指南》中“创建集群”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 若需要使用upsert模式，则必须在DWS结果表和该结果表连接的DWS表都定义主键。
- 若DWS在不同的schema中存在相同名称的表，则在flink opensource sql中需要指定相应的schema。
- 提交Flink作业前，建议勾选“保存作业日志”参数，在OBS桶选项中选择日志保存的位置，方便后续作业提交失败或运行异常时，查看日志并分析问题原因。
- 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。该驱动为默认，创建表时可以不填该驱动参数。

 版权所有 © 华为技术有限公司

例如，使用gsjdbc4驱动连接、upsert模式写入数据到DWS中。

```
create table dwsSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://DwsAddress:DwsPort/DwsDatabase',
  'table-name' = 'car_info',
  'username' = 'DwsUserName',
  'password' = 'DwsPasswrod',
  'write.mode' = 'upsert'
);
```

- 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。

  当DWS表test在名为ads_game_sdk_base的schema下时，可以参考如下样例创建DWS结果表。

```
create table dwsSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector' = 'gaussdb',
  'table-name' = 'ads_game_sdk_base\".\"test',
  'driver' = 'com.huawei.gauss200.jdbc.Driver',
  'url' = 'jdbc:gaussdb://DwsAddress:DwsPort/DwsDatabase',
  'username' = 'DwsUserName',
  'password' = 'DwsPasswrod',
  'write.mode' = 'upsert'
);
```

语法格式

说明

DWS结果表中不允许指定所有属性为PRIMARY KEY。

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'gaussdb',
  'url' = '',
  'table-name' = '',
  'driver' = '',
  'username' = '',
  'password' = ''
);
```

参数说明

表 3-17 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定要使用的连接器，这里是'gaussdb'

 版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
url	是	无	String	jdbc连接地址 。 使用gsjdbc4驱动连接时，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。 使用gsjdbc200驱动连接时，格式为：jdbc:gaussdb://${ip}:${port}/${dbName}。
table-name	是	无	String	操作的表名。如果该DWS表在某schema下，则格式为：'schema\".\"具体表名'，具体可以参考**常见问题**说明。
driver	否	org.postgresql.Driver	String	jdbc连接驱动，默认为:org.postgresql.Driver。 • 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。 • 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。
username	否	无	String	DWS数据库认证用户名，需要和'password'一起配置
password	否	无	String	DWS数据库认证密码，需要和'username'一起配置
write.mode	否	无	String	数据写入模式，支持: copy, insert以及upsert三种。默认值为upsert。 该参数与'primary key'配合使用。 • 未配置'primary key'时，支持copy及insert两种模式追加写入。 • 配置'primary key'，支持copy、upsert以及insert三种模式更新写入。 注意：由于dws不支持更新分布列，因而配置的更新主键必须包含dws表中定义的所有分布列。
sink.buffer-flush.max-rows	否	100	Integer	每次写入请求缓存的最大行数。 它能提升写入数据的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
sink.buffer-flush.interval	否	1s	Duration	刷新缓存的间隔，在这段时间内以异步线程刷新数据。 它能提升写入数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。 注意："sink.buffer-flush.max-size" 和 "sink.buffer-flush.max-rows" 同时设置为 "0"，并设置刷新缓存的间隔，则以完整的异步处理方式刷新缓存。 格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
sink.max-retries	否	3	Integer	写入最大重试次数。
write.escape-string-value	否	false	Boolean	是否对string类型值进行转义。该参数仅用于write.mode为copy模式下。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置置账号和密码。
key-by-before-sink	否	false	Boolean	在sink算子前是否按指定的主键进行分区。 该参数旨在解决多并发写入的场景下且write.mode为upsert时，如果多个子任务中写入sink的一批数据具有不止一条相同的主键，并且主键相同的这些数据先后顺序不一致，就会导致两个子任务在向DWS根据主键获取行锁时发生互锁的问题。

示例

该示例是从kafka数据源中读取数据，并以insert模式写入DWS结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，在DLI上根据DWS和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置DWS和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据DWS和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 连接DWS数据库，在DWS中创建相应的表，表名为dws_order，SQL语句参考如下：

```
create table public.dws_order(
  order_id VARCHAR,
  order_channel VARCHAR,
```


版权所有 © 华为技术有限公司

```
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR);
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将Kafka作业数据源，将DWS作为结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE dwsSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://***DWSAddress:DWSPort/DWSdbName***',
  'table-name' = 'dws_order',
  'driver' = 'org.postgresql.Driver',
  'username' = '***DWSUserName***',
  'password' = '***DWSPassword***',
  'write.mode' = 'insert'
);

insert into dwsSink select * from kafkaSource;
```

5. 连接Kafka集群，向Kafka中输入以下测试数据。

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

6. 从DWS中使用如下SQL语句查看数据结果。

```
 select * from dws_order
```

   数据结果参考如下：

```
202103241000000001    webShop    2021-03-24 10:00:00    100.0    100.0    2021-03-24 10:02:03    0001    Alice    330106
```


版权所有 © 华为技术有限公司

常见问题

- Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

  A：应考虑是跨源没有绑定，或者跨源没有绑定成功。

  - 参考**增强型跨源连接**章节，重新配置跨源。参考**DLI跨源连接DWS失败**进行问题排查。

- Q：如果该DWS表在某schema下，则应该如何配置?

  A：当DWS表test在名为ads_game_sdk_base的schema下时，可以参考如下样例中的'table-name'参数配置。

```
CREATE TABLE ads_rpt_game_sdk_realtime_ada_reg_user_pay_mm (
  ddate DATE,
  dmin TIMESTAMP(3),
  game_appkey VARCHAR,
  channel_id VARCHAR,
  pay_user_num_1m bigint,
  pay_amt_1m bigint,
  PRIMARY KEY (ddate, dmin, game_appkey, channel_id) NOT ENFORCED
) WITH (
  'connector' = 'gaussdb',
  'url' = 'jdbc:postgresql://<yourDwsAddress>:<yourDwsPort>/dws_bigdata_db',
  'table-name' = 'ads_game_sdk_base\".\"test',
  'username' = '<yourUsername>',
  'password' = '<yourPassword>',
  'write.mode' = 'upsert'
);
```

- Q：作业运行正常，但是DWS中一直没有数据怎么办?

  A：请分别排查以下场景：

  - 查看jobmanager和taskmanager的日志是否有错误抛出。日志查看操作步骤如下：
    i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
    ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
    iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”或“jobmanager”的文件夹进入，下载获取taskmanager.out和jobmanager.out文件查看结果日志。
  - 验证跨源是否正确绑定且安全组规则已对该队列开放。
  - 查看所要写入的DWS表是否在多个不同的schema中存在。若存在，则需要在flink作业中指定schema。

3.3.2.4 Elasticsearch 结果表

功能描述

DLI将Flink作业的输出数据输出到云搜索服务CSS的Elasticsearch中。Elasticsearch是基于Lucene的当前流行的企业级搜索服务器，具备分布式多用户的能力。其主要功能包括全文检索、结构化搜索、分析、聚合、高亮显示等。能为用户提供实时搜索、稳定可靠的服务。适用于日志分析、站内搜索等场景。

版权所有 © 华为技术有限公司

云搜索服务（Cloud Search Service，简称CSS）为DLI提供托管的分布式搜索引擎服务，完全兼容开源Elasticsearch搜索引擎，支持结构化、非结构化文本的多条件检索、统计、报表。

云搜索服务的更多信息，请参见**《云搜索服务用户指南》**

前提条件

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 请务必确保您的账户下已在云搜索服务里创建了集群。如何创建集群请参考《云搜索服务用户指南》中**创建集群**章节。
- 该场景作业需要运行在DLI的独享队列上，因此要与云搜索服务建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 当前只支持CSS集群7.X及以上版本，推荐使用7.6.2版本。
- 若未开启安全模式，无需使用任何跨源认证，即无需配置pwd_auth_name、es_auth_name、user_name、password、certificate，且语法中hosts字段值以http开头。
- 若开启安全模式，未开启https：
  - 方法1：推荐使用password类型跨源认证，并配置pwd_auth_name为跨源认证的名称，且语法中hosts字段值以http开头。
  - 方法2：不使用跨源认证，但需要配置用户名username、密码password，且语法中hosts字段值以http开头。
- 若开启安全模式，开启https：
  - 方法1：推荐使用CSS类型跨源认证名称，并配置es_auth_name为跨源认证的名称。请注意该场景hosts字段值以https开头。
  - 方法2：不使用跨源认证，但需要配置用户名username、密码password、证书位置certificate。请注意该场景hosts字段值以https开头。
- CSS集群安全组入向规则必须开启ICMP。
- 数据类型的使用，请参考**Format**章节。
- 提交Flink作业前，建议勾选“保存作业日志”参数，在OBS桶选项中选择日志保存的位置，方便后续作业提交失败或运行异常时，查看日志并分析问题原因。
- Elasticsearch结果表根据是否定义了主键确定是在upsert模式还是在append模式下工作。
  - 如果定义了主键，Elasticsearch Sink将在upsert模式下工作，该模式可以消费包含UPDATE和DELETE的消息。
  - 如果未定义主键，Elasticsearch Sink将以append模式工作，该模式只能消费INSERT消息。

版权所有 © 华为技术有限公司

在Elasticsearch结果表中，主键用于计算Elasticsearch的文档ID。文档ID为最多512个字节不包含空格的字符串。Elasticsearch结果表通过使用“document-id.key-delimiter”参数指定的键分隔符按照DDL中定义的顺序连接所有主键字段，从而为每一行生成一个文档ID字符串。某些类型（例如BYTES、ROW、ARRAY和MAP等）由于没有对应的字符串表示形式，所以不允许其作为主键字段。如果未指定主键，Elasticsearch将自动生成随机的文档ID。

- Elasticsearch结果表同时支持静态索引和动态索引。
  - 如果使用静态索引，则索引选项值应为纯字符串，例如myusers，所有记录都将被写入myusers索引。
  - 如果使用动态索引，可以使用{field_name}引用记录中的字段值以动态生成目标索引。您还可以使用 {field_name|date_format_string}将TIMESTAMP、DATE和TIME类型的字段值转换为date_format_string指定的格式。date_format_string与Java的DateTimeFormatter兼容。例如，如果设置为myusers-{log_ts|yyyy-MM-dd}，则log_ts字段值为2020-03-27 12:25:55的记录将被写入myusers-2020-03-27索引。

语法格式

```
create table esSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'elasticsearch-7',
  'hosts' = '',
  'index' = ''
);
```

参数说明

表 3-18 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定要使用的连接器，固定为：elasticsearch-7。表示连接到 Elasticsearch 7.x 及更高版本集群。
hosts	是	无	String	Elasticsearch所在集群的主机名，多个以';'间隔。
index	是	无	String	每条记录的 Elasticsearch 索引。可以是静态索引（例如'myIndex'）或动态索引（例如'index-{log_ts|yyyy-MM-dd}'）。
username	否	无	String	Elasticsearch所在集群的账号。该账号参数需和密码“password”参数同时配置。
password	否	无	String	Elasticsearch所在集群的密码。该密码参数需和“username”参数同时配置。

 版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
certificate	否	无	String	Elasticsearch集群的证书在obs中的位置。 例如：obs://bucket/path/CloudSearchService.cer 仅在开启安全模式，且开启https，且未使用其他跨源认证的场景下下需要配置该参数。
document-id.key-delimiter	否	_	String	连接复合主键的拼接符，默认为_。
failure-handler	否	fail	String	对Elasticsearch请求失败时的故障处理策略。有效的策略是： ● fail: 如果请求失败并因此导致作业失败，则抛出异常。 ● ignore: 忽略失败并丢弃请求。 ● retry-rejected：重新添加由于队列容量饱和而失败的请求。 ● 自定义类名：用于使用ActionRequestFailureHandler子类进行故障处理。
sink.flush-on-checkpoint	否	true	Boolean	是否在检查点刷新。 如果配置为false，在Elasticsearch进行Checkpoint时，connector将不等待确认所有pending请求已完成。因此，connector不会为请求提供at-least-once保证。
sink.bulk-flush.max-actions	否	1000	Interger	每个批量请求的最大缓冲操作数。可以设置'0'为禁用它。
sink.bulk-flush.max-size	否	2mb	MemorySize	每个批量请求的缓冲操作的内存中的最大大小。必须是MB粒度。可以设置'0'为禁用它。
sink.bulk-flush.interval	否	1s	Duration	刷新缓冲操作的间隔。可以设置'0'为禁用它。 请注意: 'sink.bulk-flush.max-size'和'sink.bulk-flush.max-actions' 都可以设置为'0'刷新间隔，从而允许对缓冲操作进行完整的异步处理。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
sink.bulk-flush.backoff.strategy	否	DISABLED	String	指定在任何刷新操作由于临时请求错误而失败时如何执行重试。有效的策略是： • DISABLED：未执行重试，即在第一个请求错误后失败。 • CONSTANT：等待重试之间的退避延迟。 • EXPONENTIAL：最初等待退避延迟并在重试之间呈指数增加。
sink.bulk-flush.backoff.max-retries	否	8	Integer	最大退避重试次数。
sink.bulk-flush.backoff.delay	否	50ms	Duration	每次退避尝试之间的延迟。 对于CONSTANT退避，这只是每次重试之间的延迟。 对于EXPONENTIAL退避，这是初始基本延迟。
connection.max-retry-timeout	否	无	Duration	重试之间的最大超时时间。
connection.path-prefix	否	无	String	要添加到每个REST通信的前缀字符串，例如， '/v1'。
format	否	json	String	Elasticsearch连接器支持指定格式。该格式必须生成有效的 json 文档。默认情况下使用内置'json'格式。 请参考**Format**页面以获取更多详细信息和格式参数。
pwd_auth_name	否	无	String	Password类型的跨源认证名称。 • 仅在使用CSS类型的跨源认证时配置该参数。 • es_auth_name和pwd_auth_name只能配置一个。
es_auth_name	否	无	String	CSS类型的跨源认证的名称。 • 仅在使用CSS类型的跨源认证时配置该参数。 • es_auth_name和pwd_auth_name只能配置一个。

版权所有 © 华为技术有限公司

示例

该示例是从Kafka数据源中读取数据，并写入到Elasticsearch结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，在DLI上根据Elasticsearch和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置Elasticsearch和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据Elasticsearch和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 登录Elasticsearch集群的Kibana，并选择Dev Tools，输入下列语句并执行，以创建值为orders的index：

```
PUT /orders
{
  "settings": {
    "number_of_shards": 1
  },
    "mappings": {
      "properties": {
        "order_id": {
          "type": "text"
        },
        "order_channel": {
          "type": "text"
        },
        "order_time": {
          "type": "text"
        },
        "pay_amount": {
          "type": "double"
        },
        "real_pay": {
          "type": "double"
        },
        "pay_time": {
          "type": "text"
        },
        "user_id": {
          "type": "text"
        },
        "user_name": {
          "type": "text"
        },
        "area_id": {
          "type": "text"
        }
      }
    }
}
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
```


版权所有 © 华为技术有限公司

```
) WITH (
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE elasticsearchSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'elasticsearch-7',
  'hosts' = 'ElasticsearchAddress:ElasticsearchPort',
  'index' = 'orders'
);

insert into elasticsearchSink select * from kafkaSource;
```

5. 连接Kafka集群，向kafka中插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

6. 在Elasticsearch集群的Kibana中输入下述语句并查看相应结果：

```
GET orders/_search
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "orders",
        "_type" : "_doc",
        "_id" : "ae7wpH4B1dV9conjpXeB",
        "_score" : 1.0,
        "_source" : {
          "order_id" : "202103241000000001",
          "order_channel" : "webShop",
          "order_time" : "2021-03-24 10:00:00",
          "pay_amount" : 100.0,
          "real_pay" : 100.0,
          "pay_time" : "2021-03-24 10:02:03",
          "user_id" : "0001",
          "user_name" : "Alice",
          "area_id" : "330106"
        }
```

 版权所有 © 华为技术有限公司

```
    },
    {
      "_index" : "orders",
      "_type" : "_doc",
      "_id" : "au7xpH4B1dV9conjn3er",
      "_score" : 1.0,
      "_source" : {
        "order_id" : "202103241606060001",
        "order_channel" : "appShop",
        "order_time" : "2021-03-24 16:06:06",
        "pay_amount" : 200.0,
        "real_pay" : 180.0,
        "pay_time" : "2021-03-24 16:10:06",
        "user_id" : "0001",
        "user_name" : "Alice",
        "area_id" : "330106"
      }
    }
  ]
 }
}
```

3.3.2.5 Hbase 结果表

功能描述

DLI将作业的输出数据输出到HBase中。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。HBase支持消息数据、报表数据、推荐类数据、风控类数据、日志数据、订单数据等结构化、半结构化的KeyValue数据存储。 利用DLI，用户可方便地将海量数据高速、低时延写入HBase。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。
- 若使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机IP信息。

  详细操作请参考《 数据湖探索用户指南 》中的“**修改主机信息**”章节描述。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 创建的HBase结果表的列簇必须定义为ROW类型，字段名对应列簇名（column family），嵌套的字段名对应列限定符名（column qualifier）。用户只需在表结构中声明查询中使用的的列簇和列限定符。除了ROW类型的列，剩下的原子数据类型字段（比如，STRING, BIGINT）将被识别为 HBase的rowkey，一张表中只能

版权所有 © 华为技术有限公司

声明一个rowkey。rowkey字段的名字可以是任意的，如果是保留关键字，需要用反引号。

语法格式

```
create table hbaseSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  ','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
) with (
  'connector' = 'hbase-2.2',
  'table-name' = '',
  'zookeeper.quorum' = ''
);
```

参数说明

表 3-19 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定使用的连接器，固定为：hbase-2.2。
table-name	是	无	String	连接的HBase表名。
zookeeper.quorum	是	无	String	HBase Zookeeper实例信息，格式为：ZookeeperAddress:ZookeeperPort 以MRS Hbase集群为例，该参数的所使用Zookeeper的ip地址和端口号获取方式如下： ● 在MRS Manager上，选择“集群 > *待操作的集群名称* > 服务 > ZooKeeper > 实例”，获取ZooKeeper角色实例的IP地址。 ● 在MRS Manager上，选择“集群 > 待操作的集群名称 > 服务 > ZooKeeper > 配置 > 全部配置”，搜索参数“clientPort”，获取“clientPort”的参数值即为ZooKeeper的端口。
zookeeper.znode.parent	否	/hbase	String	Zookeeper中的根目录，默认是/hbase。
null-string-literal	否	null	String	当字符串值为null时的存储形式，默认存成"null" 字符串。 HBase sink的编解码将所有数据类型（除字符串外）为null值时以空字节来存储。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
sink.buffer-flush.max-size	否	2mb	MemorySize	每次写入请求缓存行的最大值。 它能提升写入HBase数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。
sink.buffer-flush.max-rows	否	1000	Integer	每次写入请求缓存的最大行数。 它能提升写入HBase数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。
sink.buffer-flush.interval	否	1s	Duration	刷新缓存的间隔，在这段时间内以异步线程刷新数据。 它能提升写入HBase数据库的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。 注意："sink.buffer-flush.max-size" 和 "sink.buffer-flush.max-rows" 同时设置为 "0"，并设置刷新缓存的间隔，则以完整的异步处理方式刷新缓存。 格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
sink.parallelism	否	无	Integer	为 HBase sink operator 定义并行度。 默认情况下，并行度由框架决定，和连接在一起的上游operator一样。
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。 使用跨源认证则无需在作业中置账号密码。

数据类型映射

HBase以字节数组存储所有数据。在读和写过程中要序列化和反序列化数据。

Flink 的 HBase 连接器利用 HBase（Hadoop) 的工具类 org.apache.hadoop.hbase.util.Bytes进行字节数组和Flink 数据类型转换。

Flink 的 HBase 连接器将所有数据类型（除字符串外）null值编码成空字节。对于字符串类型，null值的字面值由null-string-literal选项值决定。

版权所有 © 华为技术有限公司

表 3-20 数据类型映射表

Flink 数据类型	HBase 转换
CHAR / VARCHAR / STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY / VARBINARY	返回 byte[]。
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
SMALLINT	byte[] toBytes(short val) short toShort(byte[] bytes)
INT	byte[] toBytes(int val) int toInt(byte[] bytes)
BIGINT	byte[] toBytes(long val) long toLong(byte[] bytes)
FLOAT	byte[] toBytes(float val) float toFloat(byte[] bytes)
DOUBLE	byte[] toBytes(double val) double toDouble(byte[] bytes)
DATE	从 1970-01-01 00:00:00 UTC 开始的天数，int 值。
TIME	从 1970-01-01 00:00:00 UTC 开始天的毫秒数，int 值。
TIMESTAMP	从 1970-01-01 00:00:00 UTC 开始的毫秒数，long 值。
ARRAY	不支持
MAP / MULTISET	不支持
ROW	不支持

示例

该示例是从Kafka数据源中读取数据，并写入到HBase结果表中，其具体步骤如下（该示例中hbase的版本为1.3.1和2.2.3）：

 版权所有 © 华为技术有限公司

1. 参考**增强型跨源连接**，在DLI上根据HBase和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置HBase和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据HBase和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**MRS HBase的使用**，通过HBase shell在HBase中创建相应的表，表名为order，表中只有一个列族detail，创建语句如下：

   ```
   create 'order', {NAME => 'detail'}
   ```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将Kafka作为数据源，HBase作为结果表（Rowkey为order_id，列簇名为detail）

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   CREATE TABLE orders (
     order_id string,
     order_channel string,
     order_time string,
     pay_amount double,
     real_pay double,
     pay_time string,
     user_id string,
     user_name string,
     area_id string
   ) WITH (
     'connector' = 'kafka',
     'topic' = '***KafkaTopic***',
     'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
     'properties.group.id' = '***GroupId***',
     'scan.startup.mode' = 'latest-offset',
     'format' = 'json'
   );

   create table hbaseSink(
     order_id string,
     detail Row(
       order_channel string,
       order_time string,
       pay_amount double,
       real_pay double,
       pay_time string,
       user_id string,
       user_name string,
       area_id string)
   ) with (
     'connector' = 'hbase-2.2',
     'table-name' = 'order',
     'zookeeper.quorum' = '***ZookeeperAddress:ZookeeperPort***',
     'sink.buffer-flush.max-rows' = '1'
   );

   insert into hbaseSink select order_id,
   Row(order_channel,order_time,pay_amount,real_pay,pay_time,user_id,user_name,area_id) from orders;
   ```

5. 连接Kafka集群，向Kafka中输入数据：

   ```
   {"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
   "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
   "user_name":"Alice", "area_id":"330106"}

   {"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
   "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
   "user_name":"Alice", "area_id":"330106"}
   ```


版权所有 © 华为技术有限公司

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

6. 通过HBase shell使用下述语句查看数据结果：

```
scan 'order'
```

数据结果参考如下：

```
202103241000000001    column=detail:area_id, timestamp=2021-12-16T21:30:37.954, value=330106

202103241000000001    column=detail:order_channel, timestamp=2021-12-16T21:30:37.954, value=webShop

202103241000000001    column=detail:order_time, timestamp=2021-12-16T21:30:37.954, value=2021-03-24 10:00:00

202103241000000001    column=detail:pay_amount, timestamp=2021-12-16T21:30:37.954, value=@Y\x00\x00\x00\x00\x00\x00

202103241000000001    column=detail:pay_time, timestamp=2021-12-16T21:30:37.954, value=2021-03-24 10:02:03

202103241000000001    column=detail:real_pay, timestamp=2021-12-16T21:30:37.954, value=@Y\x00\x00\x00\x00\x00\x00

202103241000000001    column=detail:user_id, timestamp=2021-12-16T21:30:37.954, value=0001

202103241000000001    column=detail:user_name, timestamp=2021-12-16T21:30:37.954, value=Alice

202103241606060001    column=detail:area_id, timestamp=2021-12-16T21:30:44.842, value=330106

202103241606060001    column=detail:order_channel, timestamp=2021-12-16T21:30:44.842, value=appShop

202103241606060001    column=detail:order_time, timestamp=2021-12-16T21:30:44.842, value=2021-03-24 16:06:06

202103241606060001    column=detail:pay_amount, timestamp=2021-12-16T21:30:44.842, value=@i\x00\x00\x00\x00\x00\x00

202103241606060001    column=detail:pay_time, timestamp=2021-12-16T21:30:44.842, value=2021-03-24 16:10:06

202103241606060001    column=detail:real_pay, timestamp=2021-12-16T21:30:44.842, value=@f\x80\x00\x00\x00\x00\x00

202103241606060001    column=detail:user_id, timestamp=2021-12-16T21:30:44.842, value=0001

202103241606060001    column=detail:user_name, timestamp=2021-12-16T21:30:44.842, value=Alice

202103251202020001    column=detail:area_id, timestamp=2021-12-16T21:30:52.181, value=330110

202103251202020001    column=detail:order_channel, timestamp=2021-12-16T21:30:52.181, value=miniAppShop

202103251202020001    column=detail:order_time, timestamp=2021-12-16T21:30:52.181, value=2021-03-25 12:02:02

202103251202020001    column=detail:pay_amount, timestamp=2021-12-16T21:30:52.181, value=@N\x00\x00\x00\x00\x00\x00

202103251202020001    column=detail:pay_time, timestamp=2021-12-16T21:30:52.181, value=2021-03-25 12:03:00

202103251202020001    column=detail:real_pay, timestamp=2021-12-16T21:30:52.181, value=@N\x00\x00\x00\x00\x00\x00

202103251202020001    column=detail:user_id, timestamp=2021-12-16T21:30:52.181, value=0002

202103251202020001    column=detail:user_name, timestamp=2021-12-16T21:30:52.181, value=Bob
```

常见问题

Q：Flink作业运行失败，作业运行日志中如下报错信息，应该怎么解决?

```
org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard from server in 90069ms for connection id 0x0
```

A：可能是跨源连接未绑定或跨源绑定失败。参考**增强型跨源连接**重新配置跨源，Kafka集群安全组放通DLI队列的网段地址。

3.3.2.6 JDBC 结果表

功能描述

DLI通过JDBC结果表将Flink作业的输出数据输出到关系型数据库中。

前提条件

- DLI要与实例建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 如果JDBC结果表定义了主键，则连接器以upsert模式运行，否则，连接器以Append模式运行。
  - upsert模式：Flink会根据主键插入新行或更新现有行，Flink可以通过这种方式保证幂等性。为保证输出结果符合预期，建议为表定义主键。
  - Append模式：Flink 会将所有记录解释为INSERT消息，如果底层数据库发生主键或唯一约束违规，INSERT操作可能会失败。

语法格式

```
create table jdbcSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'jdbc',
  'url' = '',
  'table-name' = '',
  'driver' = '',
  'username' = '',
  'password' = ''
);
```

 版权所有 © 华为技术有限公司

参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	指定要使用的连接器，这里应该是'jdbc'。
url	是	无	String	数据库的URL。
table-name	是	无	String	读取数据库中的数据所在的表名。
driver	否	无	String	连接数据库所需要的驱动。若未配置，则会自动通过URL提取。
username	否	无	String	数据库认证用户名，需要和'password'一起配置。
password	否	无	String	数据库认证密码，需要和'username'一起配置。
sink.buffer-flush.max-rows	否	100	Integer	每次写入请求缓存的最大行数。 它能提升写入数据的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。
sink.buffer-flush.interval	否	1s	Duration	刷新缓存的间隔，在这段时间内以异步线程刷新数据。 它能提升写入数据的性能，但是也可能增加延迟。 设置为 "0" 关闭此选项。 注意："sink.buffer-flush.max-rows" 设置为 "0"，并设置刷新缓存间隔，则以完整的异步处理方式刷新缓存。 格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
sink.max-retries	否	3	Integer	将记录写入数据库失败时的最大重试次数。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置账号和密码。

版权所有 © 华为技术有限公司

数据类型映射

表 3-21 数据类型映射

MySQL类型	PostgreSQL类型	Flink SQL类型
TINYINT	-	TINYINT
SMALLINT TINYINT UNSIGNED	SMALLINT INT2 SMALLSERIAL SERIAL2	SMALLINT
INT MEDIUMINT SMALLINT UNSIGNED	INTEGER SERIAL	INT
BIGINT INT UNSIGNED	BIGINT BIGSERIAL	BIGINT
BIGINT UNSIGNED	-	DECIMAL(20, 0)
BIGINT	BIGINT	BIGINT
FLOAT	REAL FLOAT4	FLOAT
DOUBLE DOUBLE PRECISION	FLOAT8 DOUBLE PRECISION	DOUBLE
NUMERIC(p, s) DECIMAL(p, s)	NUMERIC(p, s) DECIMAL(p, s)	DECIMAL(p, s)
BOOLEAN TINYINT(1)	BOOLEAN	BOOLEAN
DATE	DATE	DATE
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]

版权所有 © 华为技术有限公司

MySQL类型	PostgreSQL类型	Flink SQL类型
CHAR(n) VARCHAR(n) TEXT	CHAR(n) CHARACTER(n) VARCHAR(n) CHARACTER VARYING(n) TEXT	STRING
BINARY VARBINARY BLOB	BYTEA	BYTES
-	ARRAY	ARRAY

示例

使用Kafka发送数据，通过JDBC结果表将Kafka数据再输出到MySQL数据库中。

1. 参考**增强型跨源连接**，在DLI上根据MySQL和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置MySQL和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据MySQL和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 登录MySQL，并使用下述命令在flink库下创建orders表。

```
CREATE TABLE `flink`.`orders` (
  `order_id` VARCHAR(32) NOT NULL,
  `order_channel` VARCHAR(32) NULL,
  `order_time` VARCHAR(32) NULL,
  `pay_amount` DOUBLE UNSIGNED NOT NULL,
  `real_pay` DOUBLE UNSIGNED NULL,
  `pay_time` VARCHAR(32) NULL,
  `user_id` VARCHAR(32) NULL,
  `user_name` VARCHAR(32) NULL,
  `area_id` VARCHAR(32) NULL,
  PRIMARY KEY (`order_id`)
)  ENGINE = InnoDB
  DEFAULT CHARACTER SET = utf8mb4
  COLLATE = utf8mb4_general_ci;
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
```

 版权所有 © 华为技术有限公司

```
  'connector' = 'kafka',
  'topic' = 'KafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE jdbcSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://MySQLAddress:MySQLPort/flink',--其中url中的flink表示MySQL中orders表所在的数据库名
  'table-name' = 'orders',
  'username' = 'MySQLUsername',
  'password' = 'MySQLPassword',
  'sink.buffer-flush.max-rows' = '1'
);

insert into jdbcSink select * from kafkaSource;
```

5. 连接Kafka集群，向Kafka相应的topic中发送如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

6. 查看表中数据，在MySQL中执行sql查询语句。

```
select * from orders;
```

其结果参考如下（注意，以下数据为从MySQL中复制的结果，并不是MySQL中的数据样式）。

```
202103241000000001,webShop,2021-03-24 10:00:00,100.0,100.0,2021-03-24 10:02:03,0001,Alice,330106
202103241606060001,appShop,2021-03-24 16:06:06,200.0,180.0,2021-03-24 16:10:06,0001,Alice,330106
```

常见问题

无

3.3.2.7 Kafka 结果表

功能描述

DLI通过Kafka结果表将Flink作业的输出数据输出到Kafka中。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

 版权所有 © 华为技术有限公司

前提条件

- 确保已创建kafka集群。
- 该场景作业需要运行在DLI的独享队列上，因此要与Kafka集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 数据类型的使用，请参考**Format**章节。

语法格式

```
create table kafkaSink(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'kafka',
  'topic' = '',
  'properties.bootstrap.servers' = '',
  'format' = ''
);
```

参数说明

表 3-22 参数说明

参数	是否必选	默认参数	数据类型	说明
connector	是	无	string	固定值为：kafka。
topic	是	无	string	结果表对应topic名称。
properties.bootstrap.servers	是	无	string	Kafka Broker地址。格式为：host:port,host:port,host:port，以英文逗号（,）分隔。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
format	是	无	string	Flink Kafka Connector在序列化来自Kafka的消息时使用的格式。该选项与'value.format'只能配置其中一个。 格式取值如下： • csv • json • avro 请参考**Format**页面以获取更多详细信息和格式参数。
topic-pattern	否	无	String	匹配读取kafka topic名称的正则表达式。 注意：“topic-pattern”和“topic”只能选择一个，不可同时存在。 例如：'topic.*' '(topic-c|topic-d)' '(topic-a|topic-b|topic-\\d*)' '(topic-a|topic-b|topic-[0-9]*)'
properties.*	否	无	String	设置和传入任意的Kafka原生配置文件。 注意： • 后缀名必须匹配在**Apache Kafka**中的配置键。 例如关闭自动创建topic：'properties.**allow.auto.create.topics**' = 'false'。 • 存在一些配置不支持配置，如'key.deserializer'和'value.deserializer'。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
key.format	否	无	String	序列化和反序列化Kafka消息key的格式。 注意： • 若配置了该参数，则'key.fields'也需要配置，否则kafka的记录中key会为空。 • 取值如下： csv json avro debezium-json canal-json maxwell-json avro-confluent raw 请参考**Format**页面以获取更多详细信息和格式参数。
key.fields	否	[]	List<String>	定义表中的列作为key的列表，同时需要配置'key.format'。 该参数默认为空，因此没有定义key。 使用形式如：'field1;field2'。
key.fields-prefix	否	无	String	为所有kafka消息键（Key）指定自定义前缀，以避免与消息体（Value）格式字段重名。
value.format	是	无	String	用于反序列化和序列化Kafka消息的值部分的格式。 注意: • format和value.format只能配置其中一个，如果同时配置两个，则会有冲突。 • 请参考**Format**页面以获取更多详细信息和格式参数。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
value.fields-include	否	ALL	枚举类型 可选值：[ALL, EXCEPT_KEY]	在解析消息体时，是否要包含消息键字段。 取值如下： ● ALL（默认值）：所有定义的字段都存放消息体（Value）解析出来的数据。 ● EXCEPT_KEY：除去key.fields定义字段，剩余的定义字段可以用来存放消息体（Value）解析出来的数据。
sink.partitioner	否	无	string	从Flink分区到Kafka分区的映射模式。映射模式的取值如下： ● fixed（默认值）：每个Flink分区对应至多一个Kafka分区。 ● round-robin：Flink分区中的数据将被轮流分配至Kafka的各个分区。 ● 自定义分区映射模式：如果fixed和round-robin不满足您的需求，您可以创建一个FlinkKafkaPartitioner的子类来自定义分区映射模式。例如org.mycompany.MyPartitioner。
sink.semantic	否	at-least-once	String	定义kafka sink的语义。 可选值为： ● at-least-once ● exactly-once ● none
sink.parallelism	否	无	Integer	定义Kafka sink算子的并行度。 默认情况下，由框架确定并行度，与上游链接算子的并行度保持一致。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
ssl_auth_name	否	无	String	DLI侧创建的Kafka_SSL类型的跨源认证名称。Kafka配置SSL时使用该配置。 注意：若仅使用SSL类型，则需要同时配置'properties.security.protocol' = 'SSL'; 若使用SASL_SSL类型，则需要同时配置'properties.security.protocol' = 'SASL_SSL'、'properties.sasl.mechanism' = 'GSSAPI或者PLAIN'、'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required username=\"xxx\" password=\"xxx\";'
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。Kafka配置SASL认证时使用该配置。 注意：如果使用SASL_PLAINTEXT类型，且使用Kerberos认证，则需要同时配置'properties.sasl.mechanism' = 'GSSAPI'和'properties.security.protocol' = 'SASL_PLAINTEXT'

示例（适用于 Kafka 集群未开启 SASL_SSL 场景）

该示例是从Kafka的一个topic中读取数据，并使用Kafka结果表将数据写入到kafka的另一个topic中。

1. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
```

 版权所有 © 华为技术有限公司

```
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE kafkaSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaSinkTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  "format" = "json"
);

insert into kafkaSink select * from kafkaSource;
```

4. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

5. 连接Kafka集群，在Kafka的sink topic读取数据，参考如下：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

示例（适用于 Kafka 集群已开启 SASL_SSL 场景）

- **示例1：DMS集群使用SASL_SSL认证方式。**

  创建DMS的kafka集群，开启SASL_SSL，并下载SSL证书，将下载的证书client.jks上传到OBS桶中。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
```

```
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks',  -- 用户上传证书的位置
  'properties.sasl.mechanism' = 'PLAIN',  -- 按照SASL_PLAINTEXT方式填写
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required
username=\"xx\" password=\"xx\";',  -- 创建kafka集群时设置的账号和密码
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = 'obs://xx/xx.jks',
  'properties.sasl.mechanism' = 'PLAIN',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required
username=\"xx\" password=\"xx\";',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **示例2：MRS集群使用kafka SASL_SSL认证方式。**
  - MRS集群请开启Kerberos认证。
  - 在”组件管理 > Kafka > 服务配置”中查找配置项” security.protocol”，并设置为”SASL_SSL”。
  - 登录MRS集群的Manager，下载用户凭据：”系统设置 > 用户管理 ，单击用户名后的”更多 > 下载认证凭据”。

    根据用户凭据生成相应的truststore.jks文件，并将用户凭据以及truststore.jks文件传入OBS中。

    具体方式请参考**客户端SSL加密功能使用说明**。
  - 若运行作业提示“Message stream modified (41)”，可能与JDK的版本有关系，可以尝试修改运行样例代码的JDK为8u_242以下版本或删除“krb5.conf”配置文件的“renew_lifetime = 0m”配置项。
  - 端口请使用KafKa服务配置中设置的sasl_ssl.port端口。
  - security.protocol请设置为SASL_SSL。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
```

```
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',  -- 用户名
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',  -- 生成truststore.jks设置的密码
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21009,xx:21009',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **示例3：MRS集群使用SASL_PAINTEXT的Kerberos认证。**
  - MRS集群请开启Kerberos认证。
  - 将“组件管理 > Kafka > 服务配置”中查找配置项” security.protocol”，并设置为”SASL_PLAINTEXT”。
  - 登录MRS集群的Manager，下载用户凭据“系统设置 > 用户管理”，单击用户名后的“更多 > 下载认证凭据”，并上传到OBS中。
  - 若运行提示“Message stream modified (41)”的错误，可能与JDK的版本有关系，可以尝试修改运行样例代码的JDK为8u_242以下版本或删除“krb5.conf”配置文件的“renew_lifetime = 0m”配置项。
  - 端口请使用KafKa服务配置中设置的sasl.port端口。
  - security.protocol请设置为SASL_PLAINTEXT。

```
CREATE TABLE ordersSources (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
```

 版权所有 © 华为技术有限公司

```
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21007,xx:21007',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:21007,xx:21007',
  'properties.sasl.kerberos.service.name' = 'kafka',
  'properties.connector.auth.open' = 'true',
  'properties.connector.kerberos.principal' = 'xx',
  'properties.connector.kerberos.krb5' = 'obs://xx/krb5.conf',
  'properties.connector.kerberos.keytab' = 'obs://xx/user.keytab',
  'properties.security.protocol' = 'SASL_PLAINTEXT',
  'properties.sasl.mechanism' = 'GSSAPI',
  "format" = "json"
);

insert into ordersSink select * from ordersSource;
```

- **示例4：MRS集群使用SSL方式。**
  - MRS集群请不要开启Kerberos认证。
  - 登录MRS集群的Manager，下载用户凭据：“系统设置 > 用户管理”。 单击用户名后的“更多 > 下载认证凭据”。

    根据用户凭据生成相应的truststore.jks文件，并将用户凭据以及truststore.jks文件传入OBS中。

    具体方式请参考**客户端SSL加密功能使用说明**。
  - 端口请注意使用KafKa服务配置中设置的ssl.port端口
  - security.protocol请设置为SSL。
  - ssl.mode.enable请设置为true。

```
CREATE TABLE ordersSource (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
```

```
) WITH (
  'connector' = 'kafka',
  'topic' = 'xx',
  'properties.bootstrap.servers' = 'xx:9093,xx:9093,xx:9093',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'properties.connector.auth.open' = 'true',
  'properties.ssl.truststore.location' = 'obs://xx/truststore.jks',
  'properties.ssl.truststore.password' = 'xx',  -- 生成truststore.jks时设置的密码
  'properties.security.protocol' = 'SSL',
  "format" = "json"
);

CREATE TABLE ordersSink (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into ordersSink select * from ordersSource;
```

3.3.2.8 Print 结果表

功能描述

Print connector用于将用户输出的数据打印到error文件或者taskmanager的文件中，方便用户查看，主要用于代码调试，查看输出结果。

前提条件

无。

注意事项

- Print结果表支持以下四种格式内容输出：

打印内容	条件1	条件2
标识符:任务 ID> 输出数据	需要提供前缀打印标识符，即创建Print表时在with参数中指定print-identifier。	parallelism > 1
标识符> 输出数据	需要提供前缀打印标识符，即创建Print表时在with参数中指定print-identifier。	parallelism == 1
任务 ID> 输出数据	不需要提供前缀打印标识符，即创建Print表时在with参数中不指定print-identifier。	parallelism > 1

 版权所有 © 华为技术有限公司

打印内容	条件1	条件2
输出数据	不需要提供前缀打印标识符，即创建Print表时在with参数中不指定print-identifier。	parallelism == 1

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table printSink (
  attr_name attr_type
  (',' attr_name attr_type) *
  (',' PRIMARY KEY (attr_name,...) NOT ENFORCED)
) with (
  'connector' = 'print',
  'print-identifier' = '',
  'standard-error' = ''
);
```

参数说明

表 3-23 参数说明

参数	是否必选	默认参数	数据类型	说明
connector	是	无	String	固定为：print。
print-identifier	否	无	String	配置一个标识符作为输出数据的前缀。
standard-error	否	false	Boolean	该值只能为true或false，默认为false。 • 若为true，则表示输出数据到taskmanager的error文件中。 • 若为false，则表示输出数据到taskmanager的out中。

示例

参考**创建Flink OpenSource作业**，创建flink opensource sql作业，运行如下作业脚本，通过DataGen表产生随机数据并输出到Print结果表中。

注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

```
create table dataGenSource(
  user_id string,
  amount int
) with (
```


版权所有 © 华为技术有限公司

```
  'connector' = 'datagen',
  'rows-per-second' = '1', --每秒生成一条数据
  'fields.user_id.kind' = 'random', --为字段user_id指定random生成器
  'fields.user_id.length' = '3' --限制user_id长度为3
);

create table printSink(
  user_id string,
  amount int
) with (
  'connector' = 'print'
);

insert into printSink select * from dataGenSource;
```

该作业提交后，作业状态变成“运行中”，后续您可通过如下操作查看输出结果。

- 方法一：
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
  c. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
- 方法二：若在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
  a. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
  b. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
  c. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

3.3.2.9 Redis 结果表

功能描述

DLI将Flink作业的输出数据输出到Redis中。Redis是一种支持Key-Value等多种数据结构的存储系统。可用于缓存、事件发布或订阅、高速队列等场景，提供字符串、哈希、列表、队列、集合结构直接存取，基于内存，可持久化。有关Redis的详细信息，请访问Redis官方网站**https://redis.io/**。

前提条件

- DLI要建立与Redis的增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

 版权所有 © 华为技术有限公司

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 若未在创建Redis结果表的语句中定义Redis key的字段，则会使用生成的uuid作为key。
- 若需要指定Redis中的key，则需要在flink的Redis结果表中定义主键，该主键的值即为key。
- Redis结果表若定义主键，则不能够定义复合主键，即主键只能是一个字段，不能是多个字段。
- schema-syntax取值约束：
  - 当schema-syntax为map或array时，非主键字段最多只能只有一个，且需要为相应的map或array类型。
  - 当schema-syntax为fields-scores时，非主键字段个数需要为偶数，且除主键字段外，每两个字段的第二个字段的类型需要为doule，会将该字段的值视为前一个字段的score。其示例如下：

```
CREATE TABLE redisSink (
  order_id string,
  order_channel string,
  order_time double,
  pay_amount STRING,
  real_pay double,
  pay_time string,
  user_id double,
  user_name string,
  area_id double,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'sorted-set',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields-scores'
);
```

- data-type取值约束：
  - 当data-type为string时，只能有一个非主键字段。
  - 当data-type为sorted-set，且schema-syntax为fields和array时，会使用default-score作为score。
  - 当data-type为sorted-set，且schema-syntax为map时，除主键字段外，只能有一个非主键字段，且需要为map类型，同时该字段的map的value需要为double类型，表示score，该字段的map的key表示redis的set中的值。
  - 当data-type为sorted-set，且schema-syntax为array-scores时，除主键字段外，只能有两个非主键字段，且这两个字段的类型需要为array.

    两个字段其中第一个字段类型是array表示Redis的set中的值，第二个字段类型为array<double>，表示相应索引的score。其示例如下：

```
CREATE TABLE redisSink (
  order_id string,
  arrayField Array<String>,
  arrayScore array<double>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '*RedisIP*',
  'password' = '*RedisPassword*',
```

```
  'data-type' = 'sorted-set',
  "default-score" = '3',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'array-scores'
);
```

语法格式

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name) NOT ENFORCED)
)
with (
  'connector' = 'redis',
  'host' = ''
);
```

参数说明

表 3-24 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'redis'。
host	是	无	String	redis连接地址。
port	否	6379	Integer	redis连接端口。
password	否	无	String	redis认证密码。
namespace	否	无	String	redis key的namespace。 例如设置该值为"person"，假设key为"jack"则redis中会是"person:jack"。
delimiter	否	:	String	redis的key和namespace之间的分隔符。
data-type	否	hash	String	redis的数据类型，有下列选项，与redis的数据类型相对应： • hash • list • set • sorted-set • string data-type取值约束详见**data-type取值约束**说明。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
schema-syntax	否	fields	String	redis的schema语义，包含以下值： • fields：适用于所有数据类型。fields类型是指可以设置多个字段，写入时会取每个字段的值。 • fields-scores：适用于sorted set数据类型，表示对每个字段都设置一个字段作为其独立的score。 • array：适用于list、set、sorted set数据类型 • array-scores：适用于sorted set数据类型 • map：适用于hash、sorted set数据类型。 schema-syntax取值约束详见**schema-syntax取值约束**说明。
deploy-mode	否	standalone	String	redis集群的部署模式，支持standalone、master-replica、cluster，默认standalone。 该值可参考redis集群的实例类型介绍。
retry-count	否	5	Integer	连接redis集群的尝试次数。
connection-timeout-millis	否	10000	Integer	尝试连接redis集群时的最大超时时间。
commands-timeout-millis	否	2000	Integer	等待操作完成响应的最大时间。
rebalancing-timeout-millis	否	15000	Integer	redis集群失败时的休眠时间。
default-score	否	0	Double	当data-type设置为“sorted-set”数据类型的默认score。
ignore-retraction	否	false	Boolean	是否忽略retract消息。
skip-null-values	否	true	Boolean	是否跳过null。若为false，则设置为字符串"null"。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置账号和密码。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
key-ttl-mode	否	no-ttl	String	key-ttl-mode是开启Redis sink TTL的功能参数，key-ttl-mode的限制为：no-ttl、expire-msec、expire-at-date、expire-at-timestamp。 • no-ttl：不设置过期时间。 • expire-msec：设置key多久过期，参数为long类型字符串，单位为毫秒。 • expire-at-date：设置key到某个时间点过期，参数为UTC时间。 • expire-at-timestamp：设置key到某个时间点过期，参数为时间戳。
key-ttl	否	无	String	key-ttl是key-ttl-mode的补充参数，有以下几种参数值： • 当key-ttl-mode取值为no-ttl时，不需要配置此参数。 • 当key-ttl-mode取值为expire-msec时，需要配置为可以解析成Long型的字符串。例如5000，表示5000ms后key过期。 • 当key-ttl-mode取值为expire-at-date时，需要配置为Date类型字符串，例如2011-12-03T10:15:30，表示到期时间为北京时间2011-12-03 18:15:30。 • 当key-ttl-mode取值为expire-at-timestamp时，需要配置为timestamp类型字符串，单位为毫秒。例如1679385600000，表示到期时间为2023-03-21 16:00:00。

示例

该示例是从Kafka数据源中读取数据，并写入Redis到结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，根据Redis所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Redis的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据redis的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

版权所有 © 华为技术有限公司

注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' = '<yourKafka>:<port>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
--如下redisSink表data-type为默认值hash，schema-syntax定义为fields，将order_id定义为主键，即将该字段的值作为redis的key
CREATE TABLE redisSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '<yourRedis>',
  'password' = '<yourPassword>',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields'
);

insert into redisSink select * from orders;
```

4. 连接Kafka集群，向Kafka中插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06", "pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

5. 在Redis中分别执行以下命令，查看运行结果：
   - 获取key为"202103241606060001"的结果。

     执行命令：

     ```
     HGETALL 202103241606060001
     ```

     运行结果：

     ```
     1) "user_id"
     2) "0001"
     3) "user_name"
     4) "Alice"
     5) "pay_amount"
     6) "200.0"
     7) "real_pay"
     8) "180.0"
     9) "order_time"
     ```

 版权所有 © 华为技术有限公司

```
10) "2021-03-24 16:06:06"
11) "area_id"
12) "330106"
13) "order_channel"
14) "appShop"
15) "pay_time"
16) "2021-03-24 16:10:06"
```

- 获取key为"202103241000000001"的结果。

  执行命令：

  ```
  HGETALL 202103241000000001
  ```

  运行结果：

  ```
  1) "user_id"
  2) "0001"
  3) "user_name"
  4) "Alice"
  5) "pay_amount"
  6) "100.0"
  7) "real_pay"
  8) "100.0"
  9) "order_time"
  10) "2021-03-24 10:00:00"
  11) "area_id"
  12) "330106"
  13) "order_channel"
  14) "webShop"
  15) "pay_time"
  16) "2021-03-24 10:02:03"
  ```

常见问题

- Q：当data-type为set时，最终结果数据相比输入数据个数少了是什么原因?

  A：这是因为输入数据中有重复数据，导致在Redis的set中会进行排重，因此个数变少了。

- Q：若Flink作业的日志中有如下报错信息，应该怎么解决?

  ```
  org.apache.flink.table.api.ValidationException: SQL validation failed. From line 1, column 40 to line 1, column 105: Parameters must be of the same type
  ```

  A：则考虑使用了array类型，但是array中各个字段的类型不统一，需要保持Redis中array中各个字段的类型统一。

- Q：若Flink作业的日志中有如下报错信息，应该怎么解决?

  ```
  org.apache.flink.addons.redis.core.exception.RedisConnectorException: Wrong Redis schema for 'map' syntax: There should be a key (possibly) and 1 MAP non-key column.
  ```

  A：schema-syantax为map时，在flink中的建表语句只能有一个非主键的列，且该列类型需要为map。

- Q：若Flink作业的日志中有如下报错信息，应该怎么解决?

  ```
  org.apache.flink.addons.redis.core.exception.RedisConnectorException: Wrong Redis schema for 'array' syntax: There should be a key (possibly) and 1 ARRAY non-key column.
  ```

  A：schema-syantax为array时，在flink中的建表语句只能有一个非主键的列，且该列类型需要为array。

- Q：data-type已经设置了类型，那么schema-syntax的作用是什么?

  A：schema-syntax实际是对特殊类型的处理，如对map和array类型的处理。

  - 对于fields，会对每个字段的值进行处理；对于array和map则会将该字段中的每个元素进行处理。当是fields时，会将该map或array类型的字段值直接作为一个redis中的一个value。
  - 而当是array或者map时，会将array中的每个值作为redis中的一个value，会将map中该字段的value作为redis中的value。array-scores用于sorted-set的data-type，表示使用两个array字段，第一个字段为set中的值，第二个字段

表示相应值所对应的score。fields-scores用于sorted-set的data-type，表示从定义的字段中获取score，该类型表示除主键外的奇数字段表示set中的值，该字段的下一个字段表示该字段的score，因此该字段的下一个字段需要为double类型。

- Q：当data-type为hash时，schema-syntax为fields和map的区别是什么？

  A：当使用fields时，会将flink中的字段名作为redis的hash数据类型的field，该字段对应的值作为redis的hash数据类型的value。而当使用map时，会将flink中该字段值的key作为redis的hash数据类型的field，该字段值的value作为redis hash数据类型的value。其具体示例如下：

  - 对于fields：

    i. 创建的Flink作业运行脚本如下：

    ```
    CREATE TABLE orders (
      order_id string,
      order_channel string,
      order_time string,
      pay_amount double,
      real_pay double,
      pay_time string,
      user_id string,
      user_name string,
      area_id string
    ) WITH (
      'connector' = 'kafka',
      'topic' = '***kafkaTopic***',
      'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
      'properties.group.id' = '***GroupId***',
      'scan.startup.mode' = 'latest-offset',
      'format' = 'json'
    );

    CREATE TABLE redisSink (
      order_id string,
      maptest Map<string, String>,
      primary key (order_id) not enforced
    ) WITH (
      'connector' = 'redis',
      'host' = '***RedisIP***',
      'password' = '***RedisPassword***',
      'deploy-mode' = 'master-replica',
      'schema-syntax' = 'fields'
    );

    insert into redisSink select order_id, Map[user_id, area_id] from orders;
    ```

    ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

    ```
    {"order_id":"202103241000000001", "order_channel":"webShop",
    "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
    "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
    "area_id":"330106"}
    ```

    iii. 在Redis中，查看其结果如下：

    ```
    1) "maptest"
    2) "{0001=330106}"
    ```

  - 对于map：

    i. 对于map而言，创建的Flink作业运行脚本如下：

    ```
    CREATE TABLE orders (
      order_id string,
      order_channel string,
      order_time string,
      pay_amount double,
      real_pay double,
      pay_time string,
      user_id string,
      user_name string,
    ```

```
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE redisSink (
  order_id string,
  maptest Map<string, String>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'map'
);

insert into redisSink select order_id, Map[user_id, area_id] from orders;
```

ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop",
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
"area_id":"330106"}
```

iii. 在Redis中，查看其结果如下：

```
1) "0001"
2) "330106"
```

- Q：当data-type为list时，schema-syntax为fields和array的区别是什么？

  A：fields和array的不同不会导致结果不同。只是在flink建表语句中不同，fields可以是多个字段，而array需要该字段为array类型，且array中的数据类型必须相同，因此fields会更加灵活。

  - 对于fields：

    i. 对于fields而言，创建的Flink作业运行脚本如下：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE redisSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
```

 版权所有 © 华为技术有限公司

```
  user_name string,
  area_id string,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'list',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields'
);

insert into redisSink select * from orders;
```

ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop",
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
"area_id":"330106"}
```

iii. 使用以下命令查看其结果如下：

Redis执行以下命令：

```
LRANGE 202103241000000001 0 8
```

查询命令执行结果：

```
1) "webShop"
2) "2021-03-24 10:00:00"
3) "100.0"
4) "100.0"
5) "2021-03-24 10:02:03"
6) "0001"
7) "Alice"
8) "330106"
```

- 对于array：

i. 对于array而言，创建的Flink作业运行脚本如下：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = 'kafkaTopic',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE redisSink (
  order_id string,
  arraytest Array<String>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'list',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'array'
);
```

 版权所有 © 华为技术有限公司

```
insert into redisSink select order_id,
array[order_channel,order_time,pay_time,user_id,user_name,area_id] from orders;
```

ii. 连接Kafka集群，向Kafka的topic插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop",
"order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00",
"pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice",
"area_id":"330106"}
```

iii. 在Redis中，查看其结果如下（与fields结果不同是因为这里array类型，在flink中的sink建表语句中没有加入double类型的数据，因此少了两个值，并不是由于fields与array不同导致）：

```
1) "webShop"
2) "2021-03-24 10:00:00"
3) "2021-03-24 10:02:03"
4) "0001"
5) "Alice"
6) "330106"
```

3.3.2.10 Upsert Kafka 结果表

功能描述

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。DLI将Flink作业的输出数据以upsert的模式输出到Kafka中。

Upsert Kafka 连接器支持以upsert方式从Kafka topic中读取数据并将数据写入Kafka topic。

upsert-kafka连接器作为 sink，可以消费changelog 流。它会将INSERT/UPDATE_AFTER数据作为正常的Kafka消息写入，并将DELETE数据以value为空的Kafka消息写入（表示对应 key 的消息被删除）。Flink将根据主键列的值对数据进行分区，从而保证主键上的消息有序，因此同一主键上的更新/删除消息将落在同一分区中。

前提条件

- 确保已创建Kafka集群。
- 该场景作业需要运行在DLI的独享队列上，因此要与Kafka集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 数据类型的使用，请参考**Format**章节。
- Upsert Kafka始终以upsert方式工作，并且需要在 DDL 中定义主键。

 版权所有 © 华为技术有限公司

- 默认情况下，如果启用checkpoint，Upsert Kafka sink会保证至少一次将数据插入Kafka topic。这意味着，Flink可以将具有相同key的重复记录写入Kafka topic。因此，upsert-kafka 连接器可以实现幂等写入。

语法格式

```
create table kafkaSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'upsert-kafka',
  'topic' = '',
  'properties.bootstrap.servers' = '',
  'key.format' = '',
  'value.format' = ''
);
```

参数说明

表 3-25 参数说明

参数	是否必选	默认参数	数据类型	说明
connector	是	(none)	String	connector类型，对于upsert kafka，需配置为'upsert-kafka'。
topic	是	(none)	String	Kafka topic名。
properties.bootstrap.servers	是	(none)	String	Kafka brokers地址，以逗号分隔。
key.format	是	(none)	String	用于对Kafka消息中key部分序列化和反序列化的格式。key字段由PRIMARY KEY语法指定。支持的格式如下： • csv • json • avro 请参考**Format**页面以获取更多详细信息和格式参数。
key.fields-prefix	否	(none)	String	为键格式的所有字段定义自定义前缀，以避免与值格式的字段发生名称冲突。 默认情况下，前缀为空。如果定义了自定义前缀，则表架构 和'key.fields'都将使用前缀名称。在构造密钥格式的数据类型时，将删除前缀，并在密钥格式中使用无前缀的名称。请注意，此选项要求'value.fields-include' 必须设置为'EXCEPT_KEY'。

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
value.format	是	(none)	String	用于对 Kafka 消息中 value 部分序列化和反序列化的格式。支持的格式： • csv • json • avro 请参考**Format**页面以获取更多详细信息和格式参数。
value.fields-include	否	‘ALL’	String	控制哪些字段应该出现在value中。可取值： • ALL：消息的value 部分将包含 schema 的所有字段，包括定义中键的字段。 • EXCEPT_KEY：记录的value 部分包含 schema 的所有内容，定义为主键的字段除外。
sink.parallelism	否	(none)	Interger	定义upsert-kafka sink 算子的并行度。默认情况下，由框架确定并行度，与上游链接算子的并行度保持一致。
properties.*	否	(none)	String	该选项可以传递任意的 Kafka 参数。 选项的后缀名必须匹配定义在 **kafka参数文档**中的参数名。 Flink会自动移除选项名中的 "properties." 前缀，并将转换后的键名以及值传入 KafkaClient。 例如：您可以通过 'properties.allow.auto.create.topics' = 'false' 来禁止自动创建 topic。 但是 'key.deserializer' 和 'value.deserializer' 是不允许通过该方式传递参数，因为Flink会重写这些参数的值。
ssl_auth_name	否	无	String	DLI侧创建的Kafka_SSL类型的跨源认证名称。Kafka配置SSL时使用该配置。 注意：若仅使用SSL类型，则需要同时配置'properties.security.protocol '= 'SSL'; 若使用SASL_SSL类型，则需要同时配置 'properties.security.protocol' = 'SASL_SSL'、 'properties.sasl.mechanism' = 'GSSAPI或者PLAIN'、 'properties.sasl.jaas.config' = 'org.apache.kafka.common.security.plain.PlainLoginModule required username=\"xxx\" password=\"xxx\";'

版权所有 © 华为技术有限公司

参数	是否必选	默认参数	数据类型	说明
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。Kafka配置SASL认证时使用该配置。 注意：如果使用SASL_PLAINTEXT类型，且使用Kerberos认证，则需要同时配置'properties.sasl.mechanism' = 'GSSAPI'和'properties.security.protocol' = 'SASL_PLAINTEXT'

示例

从Kafka源表获取Kafka source topic数据，通过Upsert Kafka结果表将Kafka source topic数据写入到Kafka sink topic中。

1. 参考**增强型跨源连接**，根据Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);
CREATE TABLE UPSERTKAFKASINK (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '***KafkaTopic***',
```


版权所有 © 华为技术有限公司

```
  'properties.bootstrap.servers' =  'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'key.format' = 'json',
  'value.format' = 'json'
);
insert into UPSERTKAFKASINK
select * from orders;
```

4. 连接Kafka集群，kafka中source topic发送如下测试数据：

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

5. 连接Kafka集群，获取kafka sink topic的数据，结果参考如下：

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05", "pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003", "user_name":"Cindy", "area_id":"330108"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

常见问题

无

3.3.2.11 FileSystem 结果表

功能描述

FileSystem sink用于将数据输出到分布式文件系统HDFS或者对象存储服务OBS等文件系统。适用于数据转储、大数据分析、备份或活跃归档、深度或冷归档等场景。

考虑到输入流可以是无界的，每个桶中的数据被组织成有限大小的Part文件。完全可以配置为基于时间的方式往桶中写入数据，比如可以设置每个小时的数据写入一个新桶中。即桶中将包含一个小时间隔内接收到的记录。

桶目录中的数据被拆分成多个Part文件。对于相应的接收数据的桶的Sink的每个Subtask，每个桶将至少包含一个Part文件。将根据配置的滚动策略来创建其他Part文件。对于Row Formats默认的策略是根据Part文件大小进行滚动，需要指定文件打开状态最长时间的超时以及文件关闭后的非活动状态的超时时间。对于Bulk Formats在每次创建Checkpoint时进行滚动，并且用户也可以添加基于大小或者时间等的其他条件。

说明

- 在STREAMING模式下使用FileSink需要开启Checkpoint功能。Part文件只在Checkpoint成功时生成。如果没有开启Checkpoint功能，文件将永远停留在in-progress或者pending的状态，并且下游系统将不能安全读取该文件数据。
- sink end算子的接受记录数为checkpoint的个数，非实际的发送数据，实际发送数据量请参考streaming-writer或StreamingFileWriter算子的记录数。

语法格式

```
CREATE TABLE sink_table (
  name string,
  num INT,
  p_day string,
  p_hour string
) partitioned by (p_day, p_hour) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://*** ',
  'format' = 'parquet',
  'auto-compaction' = 'true'
);
```

使用说明

- **滚动策略**

  RollingPolicy 定义了何时关闭给定的In-progress Part文件，并将其转换为Pending状态，然后再转换为Finished状态。 Finished状态的文件，可供查看并且可以保证数据的有效性，在出现故障时不会恢复。

  在 STREAMING模式下，滚动策略结合Checkpoint间隔（到下一个Checkpoint成功时，文件的Pending状态才转换为 Finished 状态），共同控制Part文件对下游readers是否可见以及这些文件的大小和数量。详见滚动策略相关**参数说明**。

- **Part文件生命周期**

  为了在下游使用 FileSink 作为输出，需要了解生成的输出文件的命名和生命周期。

  Part 文件可以处于以下三种状态中的任意一种：

  - **In-progress**：当前正在写入的 Part 文件处于 in-progress 状态
  - **Pending**：由于指定的滚动策略）关闭 in-progress 状态文件，并且等待提交
  - **Finished**：流模式(STREAMING)下的成功的 Checkpoint 或者批模式(BATCH)下输入结束，文件的Pending状态转换为 Finished 状态

  只有 Finished 状态下的文件才能被下游安全读取，并且保证不会被修改。

  默认的，Part文件命名策略如下:

  - In-progress / Pending：part-<uid>-<partFileIndex>.inprogress.uid
  - Finished：part-<uid>-<partFileIndex>

  当Sink Subtask实例化时，uid是一个分配给 Subtask 的随机ID值。uid不具有容错机制，所以当Subtask从故障恢复时，uid会重新生成。

- **文件合并**

  FileSink 开始支持已经提交Pending文件的合并，从而允许应用设置一个较小的时间周期并且避免生成大量的小文件。

  这一功能开启后，在文件转为Pending状态与文件最终提交之间会进行文件合并。这些Pending状态的文件将首先被提交为一个以.开头的临时文件。这些临时文件随后将会按照用户指定的策略和合并方式进行合并，最终生成合并后的Pending状态的文件。 然后这些文件将被发送给Committer并提交为正式文件，在这之后，原始的临时文件也会被删除掉。

- **分区功能**

  Filesystem sink支持分区功能，通过partitioned by语法根据选择的字段进行分区。示例如下：

  ```
  path
  └── datetime=2022-06-25
      └── hour=10
  ```

 版权所有 © 华为技术有限公司

```
    ├── part-0.parquet
    ├── part-1.parquet
└── datetime=2022-06-26
    └── hour=16
        ├── part-0.parquet
    └── hour=17
        ├── part-0.parquet
```

分区和文件一样，也需要进行提交，通知下游应用可以安全地读取分区内的文件。Filesystem sink提供多种提交配置策略。

参数说明

表 3-26 参数说明

参数	是否必选	默认值	类型	说明
connector	是	无	String	固定位filesystem。
path	是	无	String	OBS路径。
format	是	无	String	文件格式。 支持csv、parquet格式。
sink.rolling-policy.file-size	否	128MB	MemorySize	单个part文件最大大小，超过该数值会滚动产生新文件。 **说明** RollingPolicy 定义了何时关闭给定的In-progress Part文件，并将其转换为Pending状态，然后再转换为Finished状态。 Finished状态的文件，可供查看并且可以保证数据的有效性，在出现故障时不会恢复。 在STREAMING模式下，滚动策略结合Checkpoint间隔（到下一个Checkpoint成功时，文件的Pending状态才转换为Finished状态）共同控制Part文件对下游readers是否可见以及这些文件的大小和数量。
sink.rolling-policy.rollover-interval	否	30 min	Duration	单个Part文件处于打开状态的最长时间，超过该时间会滚动产生新文件（默认值30分钟，以避免产生大量小文件）。检查频率是通过sink.rolling-policy.check-interval参数控制的。 **说明** 该参数数字与单位之间必须要有空格。 支持的时间单位包括: d,h,min,s,ms等。 对于bulk格式的文件(parquet、orc、avro)，checkpoint的时间间隔也会控制单个part文件打开的最长时间。
sink.rolling-policy.check-interval	否	1 min	Duration	基于时间的滚动策略的检查间隔。 该属性控制了基于sink.rolling-policy.rollover-interval属性检查文件是否该被滚动的检查频率。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
auto-compaction	否	false	Boolean	在流式 sink 中是否开启自动合并功能。数据首先会被写入临时文件。当checkpoint完成后，该checkpoint产生的临时文件会被合并。
compaction.file-size	否	`sink.rolling-policy.file-size`的大小	MemorySize	合并目标文件大小，默认值为滚动文件大小。 **说明** • 只有在同个checkpoint内的文件会被合并，因此最终文件的数量至少等于checkpoint的数量。 • 如果合并时间较长，可能会引起反压，延长checkpoint所需时间。 • 开启该功能后，checkpoint时会产生最终文件，并打开新的文件接收下个checkpoint产生的数据。

示例一

使用datagen随机生成数据写入obs的bucketName桶下的fileName目录中。文件生成时间与checkpoint无关，达到30min或128MB时，生成新文件。

```
create table orders(
  name string,
  num INT
) with (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.name.kind' = 'random',
  'fields.name.length' = '5'
);

CREATE TABLE sink_table (
   name string,
   num INT
) WITH (
   'connector' = 'filesystem',
   'path' = 'obs://bucketName/fileName',
   'format' = 'csv',
   'sink.rolling-policy.file-size'='128m',
   'sink.rolling-policy.rollover-interval'='30 min'
);
INSERT into sink_table SELECT * from orders;
```

示例二

使用datagen随机生成数据写入obs的bucketName桶下的fileName目录中。文件生成时间与checkpoint有关，达到checkpoint间隔或达到100MB时，生成新文件。

```
create table orders(
  name string,
  num INT
) with (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.name.kind' = 'random',
  'fields.name.length' = '5'
```


版权所有 © 华为技术有限公司

```
);

CREATE TABLE sink_table (
  name string,
  num INT
) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://bucketName/fileName',
  'format' = 'csv',
  'sink.rolling-policy.file-size'='128m',
  'sink.rolling-policy.rollover-interval'='30 min',
  'auto-compaction'='true',
  'compaction.file-size'='100m'

);
INSERT into sink_table SELECT * from orders;
```

3.3.3 创建维表

3.3.3.1 DWS 维表

功能描述

创建DWS表用于与输入流连接，从而生成相应的宽表。

前提条件

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。如何创建DWS集群，请参考《数据仓库服务管理指南》中“**创建集群**”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector' = 'gaussdb',
  'url' = '',
  'table-name' = '',
```

```
  'username' = '',
  'password' = ''
);
```

参数说明

表 3-27 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'gaussdb'。
url	是	无	String	jdbc连接地址。 使用gsjdbc4驱动连接时，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。 使用gsjdbc200驱动连接时，格式为：jdbc:gaussdb://${ip}:${port}/${dbName}。
table-name	是	无	String	读取数据库中的数据所在的表名。
driver	否	无	String	jdbc连接驱动，默认为:org.postgresql.Driver。 • 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。 • 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。
username	否	无	String	数据库认证用户名，需要和'password'一起配置。
password	否	无	String	数据库认证密码，需要和'username'一起配置。
scan.partition.column	否	无	String	用于对输入进行分区的列名。 与scan.partition.lower-bound、scan.partition.upper-bound、scan.partition.num必须同时存在或者同时不存在。
scan.partition.lower-bound	否	无	Integer	第一个分区的最小值。 与scan.partition.column、scan.partition.upper-bound、scan.partition.num必须同时存在或者同时不存在。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
scan.partition.upper-bound	否	无	Integer	最后一个分区的最大值。 与scan.partition.column、scan.partition.lower-bound、scan.partition.num必须同时存在或者同时不存在。
scan.partition.num	否	无	Integer	分区的个数。 与scan.partition.column、scan.partition.upper-bound、scan.partition.upper-bound必须同时存在或者同时不存在。
scan.fetch-size	否	0	Integer	每次从数据库拉取数据的行数。默认值为0，表示不限制。
scan.auto-commit	否	true	Boolean	设置自动提交标志。 它决定每一个statement是否以事务的方式自动提交。
lookup.cache.max-rows	否	无	Integer	维表配置，缓存的最大行数，超过该值时，最先添加的数据将被标记为过期。 默认表示不使用该配置。
lookup.cache.ttl	否	无	Duration	维表配置，缓存超时时间，超过该时间的数据会被剔除。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。 默认表示不使用该配置。
lookup.max-retries	否	3	Integer	维表配置，数据拉取最大重试次数。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置账号和密码。

示例

从Kafka源表中读取数据，将DWS表作为维表，并将二者生成的宽表信息写入Kafka结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，在DLI上根据DWS和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置DWS和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据DWS和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。

版权所有 © 华为技术有限公司

3. 连接DWS数据库实例，在DWS中创建相应的表，作为维表，表名为area_info，SQL语句如下：

```
create table public.area_info(
  area_id VARCHAR,
  area_province_name VARCHAR,
  area_city_name VARCHAR,
  area_county_name VARCHAR,
  area_street_name VARCHAR,
  region_name VARCHAR);
```

4. 连接DWS数据库实例，向DWS维表area_info中插入测试数据，其语句如下：

```
  insert into area_info
  (area_id, area_province_name, area_city_name, area_county_name, area_street_name, region_name)
  values
  ('330102', 'a1', 'b1', 'c1', 'd1', 'e1'),
  ('330106', 'a1', 'b1', 'c2', 'd2', 'e1'),
  ('330108', 'a1', 'b1', 'c3', 'd3', 'e1'),
  ('330110', 'a1', 'b1', 'c4', 'd4', 'e1');
```

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将Kafka作为数据源，DWS作为维表，数据输出到Kafka结果表中。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaSourceTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = 'dws-order',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

--创建地址维表
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) WITH (
  'connector' = 'gaussdb',
  'driver' = 'org.postgresql.Driver',
  'url' = 'jdbc:gaussdb://***DwsAddress:DwsPort/DwsDbName***',
  'table-name' = 'area_info',
  'username' = '***DwsUserName***',
  'password' = '***DwsPassword***',
  'lookup.cache.max-rows' = '10000',
  'lookup.cache.ttl' = '2h'
);

--根据地址维表生成详细的包含地址的订单信息宽表
create table order_detail(
    order_id string,
    order_channel string,
```

 版权所有 © 华为技术有限公司

```
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  area_province_name string,
  area_city_name string,
  area_county_name string,
  area_street_name string,
  region_name string
) with (
 'connector' = 'kafka',
 'topic' = '***KafkaSinkTopic***',
 'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
 'format' = 'json'
);

insert into order_detail
  select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
        area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
        area.area_street_name, area.region_name  from orders
  left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

6. 连接Kafka集群，向kafka中source topic中插入如下测试数据：

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",
"user_name":"Cindy", "area_id":"330108"}
```

7. 连接Kafka集群，读取kafka中sink topic中数据，结果参考如下：

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_ci
ty_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}

{"order_id":"202103251505050001","order_channel":"qqShop","order_time":"2021-03-25
15:05:05","pay_amount":500.0,"real_pay":400.0,"pay_time":"2021-03-25
15:10:00","user_id":"0003","user_name":"Cindy","area_id":"330108","area_province_name":"a1","area_c
ity_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

常见问题

- Q：若Flink作业日志中有如下报错信息，应该怎么解决?

```
java.io.IOException: unable to open JDBC writer
...
Caused by: org.postgresql.util.PSQLException: The connection attempt failed.
...
Caused by: java.net.SocketTimeoutException: connect timed out
```

  A：应考虑是跨源没有绑定，或者跨源没有绑定成功。

  - 参考**增强型跨源连接**章节，重新配置跨源。参考**DLI跨源连接DWS失败**进行问题排查。

- Q：如果该DWS表在某schema下，则应该如何配置?

A：如下示例是使用schema为dbuser2下的表area_info：

```
--创建地址维表
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) WITH (
 'connector' = 'gaussdb',
 'driver' = 'org.postgresql.Driver',
 'url' = 'jdbc:postgresql://DwsAddress:DwsPort/DwsDbname',
 'table-name' = 'dbuser2.area_info',
 'username' = 'DwsUserName',
 'password' = 'DwsPassword',
 'lookup.cache.max-rows' = '10000',
 'lookup.cache.ttl' = '2h'
);
```

3.3.3.2 Hbase 维表

功能描述

创建Hbase维表用于与输入流连接生成宽表。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- 若使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机IP信息。

  详细操作请参考《数据湖探索用户指南》中的“**修改主机信息**”章节描述。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- 所有 HBase 表的列簇必须定义为ROW类型，字段名对应列簇名（column family），嵌套的字段名对应列限定符名（column qualifier）。用户只需在表结构中声明查询中使用的的列簇和列限定符。除了 ROW 类型的列，剩下的原子数据类型字段（比如，STRING, BIGINT）将被识别为 HBase 的 rowkey，一张表中只能声明一个 rowkey。rowkey 字段的名字可以是任意的，如果是保留关键字，需要用反引号。

语法格式

```
create table hbaseSource (
  attr_name attr_type
```

 版权所有 © 华为技术有限公司

```
  (',' attr_name attr_type)*
 )
with (
  'connector' = 'hbase-2.2',
  'table-name' = '',
  'zookeeper.quorum' = ''
);
```

参数说明

表 3-28 参数说明

参数	是否必选	默认值	参数类型	说明
connector	是	无	String	connector的类型，需配置为：hbase-2.2。
table-name	是	无	String	连接的HBase表名。
zookeeper.quorum	是	无	String	HBase Zookeeper quorum 信息。格式为：ZookeeperAddress:ZookeeperPort。 以MRS Hbase集群为例，该参数的所使用Zookeeper的ip地址和端口号获取方式如下： ● 在MRS Manager上，选择“集群 > *待操作的集群名称* > 服务 > ZooKeeper > 实例”，获取ZooKeeper角色实例的IP地址。 ● 在MRS Manager上，选择“集群 > 待操作的集群名称 > 服务 > ZooKeeper > 配置 > 全部配置”，搜索参数“clientPort”，获取“clientPort”的参数值即为ZooKeeper的端口。
zookeeper.znode.parent	否	/hbase	String	HBase集群的Zookeeper根目录。
lookup.async	否	false	Boolean	是否设置异步维表。
lookup.cache.max-rows	否	-1	Long	维表配置，缓存的最大行数，超过该值时，最先添加的数据将被标记为过期。 默认表示不使用该配置。
lookup.cache.ttl	否	-1	Long	维表配置，缓存超时时间，超过该时间的数据会被剔除。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。 默认表示不使用该配置。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	参数类型	说明
lookup.max-retries	否	3	Integer	维表配置，数据拉取最大重试次数。
krb_auth_name	否	无	String	DLI侧创建的Kerberos类型的跨源认证名称。

数据类型映射

HBase以字节数组存储所有数据。在读和写过程中要序列化和反序列化数据。

Flink的HBase连接器利用HBase（ Hadoop) 的工具类 org.apache.hadoop.hbase.util.Bytes 进行字节数组和 Flink 数据类型转换。

Flink的HBase连接器将所有数据类型（ 除字符串外 ）null 值编码成空字节。对于字符串类型，null 值的字面值由null-string-literal选项值决定。

表 3-29 数据类型映射表

Flink 数据类型	HBase 转换
CHAR / VARCHAR / STRING	byte[] toBytes(String s) String toString(byte[] b)
BOOLEAN	byte[] toBytes(boolean b) boolean toBoolean(byte[] b)
BINARY / VARBINARY	返回 byte[]。
DECIMAL	byte[] toBytes(BigDecimal v) BigDecimal toBigDecimal(byte[] b)
TINYINT	new byte[] { val } bytes[0] // returns first and only byte from bytes
SMALLINT	byte[] toBytes(short val) short toShort(byte[] bytes)
INT	byte[] toBytes(int val) int toInt(byte[] bytes)
BIGINT	byte[] toBytes(long val) long toLong(byte[] bytes)
FLOAT	byte[] toBytes(float val) float toFloat(byte[] bytes)
DOUBLE	byte[] toBytes(double val) double toDouble(byte[] bytes)

版权所有 © 华为技术有限公司

Flink 数据类型	HBase 转换
DATE	从 1970-01-01 00:00:00 UTC 开始的天数，int 值。
TIME	从 1970-01-01 00:00:00 UTC 开始天的毫秒数，int 值。
TIMESTAMP	从 1970-01-01 00:00:00 UTC 开始的毫秒数，long 值。
ARRAY	不支持
MAP / MULTISET	不支持
ROW	不支持

示例

该示例是从Kafka数据源中读取数据，将HBase表作为维表，从而生成宽表，并将结果写入到Kafka结果表中，其具体步骤如下（该示例中HBase的版本为1.3.1和2.2.3）：

1. 参考**增强型跨源连接**，在DLI上根据HBase和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。参考“**修改主机信息**”章节描述，在增强型跨源中增加MRS的主机信息。
2. 设置HBase和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据HBase和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 参考**MRS HBase的使用**，通过HBase shell在HBase中创建相应的表，表名为area_info，表中只有一个列族detail，创建语句如下：

   ```
   create 'area_info', {NAME => 'detail'}
   ```

4. 在HBase shell中执行下述语句，插入相应的维表数据：

   ```
   put 'area_info', '330106', 'detail:area_province_name', 'a1'
   put 'area_info', '330106', 'detail:area_city_name', 'b1'
   put 'area_info', '330106', 'detail:area_county_name', 'c2'
   put 'area_info', '330106', 'detail:area_street_name', 'd2'
   put 'area_info', '330106', 'detail:region_name', 'e1'

   put 'area_info', '330110', 'detail:area_province_name', 'a1'
   put 'area_info', '330110', 'detail:area_city_name', 'b1'
   put 'area_info', '330110', 'detail:area_county_name', 'c4'
   put 'area_info', '330110', 'detail:area_street_name', 'd4'
   put 'area_info', '330110', 'detail:region_name', 'e1'
   ```

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，并提交运行。该作业脚本将Kafka作为数据源，HBase作为维表，将数据写入到Kafka作为结果表中。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   CREATE TABLE orders (
     order_id string,
     order_channel string,
     order_time string,
     pay_amount double,
     real_pay double,
     pay_time string,
   ```


版权所有 © 华为技术有限公司

```
  user_id string,
  user_name string,
  area_id string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaSourceTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

--创建地址维表
create table area_info (
  area_id string,
  detail row(
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string)
) WITH (
  'connector' = 'hbase-2.2',
  'table-name' = 'area_info',
  'zookeeper.quorum' = '***ZookeeperAddress:ZookeeperPort***',
  'lookup.async' = 'true',
  'lookup.cache.max-rows' = '10000',
  'lookup.cache.ttl' = '2h'
);

--根据地址维表生成详细的包含地址的订单信息宽表
create table order_detail(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) with (
  'connector' = 'kafka',
  'topic' = '<yourSinkTopic>',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'format' = 'json'
);

insert into order_detail
    select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
           area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
           area.area_street_name, area.region_name  from orders
    left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

6. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

 版权所有 © 华为技术有限公司

```
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25 12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00", "user_id":"0002", "user_name":"Bob", "area_id":"330110"}
```

7. 连接Kafka集群，在Kafka的sink topic读取数据，结果数据参考如下：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24 10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24 10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_city_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24 16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24 16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_city_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25 12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25 12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_city_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}
```

常见问题

Q：Flink作业日志中有如下报错信息应该怎么解决?

```
org.apache.zookeeper.ClientCnxn$SessionTimeoutException: Client session timed out, have not heard from server in 90069ms for connection id 0x0
```

A：可能是跨源连接未绑定或跨源绑定失败。参考**增强型跨源连接**重新配置跨源，Kafka集群安全组放通DLI队列的网段地址。

3.3.3.3 JDBC 维表

创建JDBC表用于与输入流连接。

前提条件

请务必确保您的账户下已创建了相应实例。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

语法格式

```
CREATE TABLE  table_id (
  attr_name attr_type
  (',' attr_name attr_type)*
)
  WITH (
  'connector' = 'jdbc',
  'url' = '',
  'table-name' = '',
  'driver' = '',
  'username' = '',
  'password' = ''
);
```

 版权所有 © 华为技术有限公司

参数说明

表 3-30 参数说明

参数	是否必选	说明
connector	是	数据源类型，固定为：jdbc。
url	是	数据库的URL。
table-name	是	读取数据库中的数据所在的表名。
driver	否	连接数据库所需要的驱动。若未配置，则会自动通过URL提取。
username	否	数据库认证用户名，需要和'password'一起配置。
password	否	数据库认证密码，需要和'username'一起配置。
scan.partition.column	否	用于对输入进行分区的列名。 与scan.partition.lower-bound、scan.partition.upper-bound、scan.partition.num必须同时存在或者同时不存在。
scan.partition.lower-bound	否	第一个分区的最小值。 与scan.partition.column、scan.partition.upper-bound、scan.partition.num必须同时存在或者同时不存在
scan.partition.upper-bound	否	最后一个分区的最大值。 与scan.partition.column、scan.partition.lower-bound、scan.partition.num必须同时存在或者同时不存在
scan.partition.num	否	分区的个数。 与scan.partition.column、scan.partition.upper-bound、scan.partition.upper-bound必须同时存在或者同时不存在。
scan.fetch-size	否	每次从数据库拉取数据的行数。默认值为0，表示忽略该提示。
lookup.cache.max-rows	否	维表配置，缓存的最大行数，超过该值时，最先添加的数据将被标记为过期。-1表示不使用缓存。

版权所有 © 华为技术有限公司

参数	是否必选	说明
lookup.cache.ttl	否	维表配置，缓存超时时间，超过该时间的数据会被剔除。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
lookup.max-retries	否	维表配置，数据拉取最大重试次数，默认为3。
pwd_auth_name	否	DLI侧创建的Password类型的跨源认证名称。

数据类型映射

表 3-31 数据类型映射

MySQL类型	PostgreSQL类型	Flink SQL类型
TINYINT	-	TINYINT
SMALLINT TINYINT UNSIGNED	SMALLINT INT2 SMALLSERIAL SERIAL2	SMALLINT
INT MEDIUMINT SMALLINT UNSIGNED	INTEGER SERIAL	INT
BIGINT INT UNSIGNED	BIGINT BIGSERIAL	BIGINT
BIGINT UNSIGNED	-	DECIMAL(20, 0)
BIGINT	BIGINT	BIGINT
FLOAT	REAL FLOAT4	FLOAT
DOUBLE DOUBLE PRECISION	FLOAT8 DOUBLE PRECISION	DOUBLE
NUMERIC(p, s) DECIMAL(p, s)	NUMERIC(p, s) DECIMAL(p, s)	DECIMAL(p, s)
BOOLEAN TINYINT(1)	BOOLEAN	BOOLEAN
DATE	DATE	DATE

版权所有 © 华为技术有限公司

MySQL类型	PostgreSQL类型	Flink SQL类型
TIME [(p)]	TIME [(p)] [WITHOUT TIMEZONE]	TIME [(p)] [WITHOUT TIMEZONE]
DATETIME [(p)]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]	TIMESTAMP [(p)] [WITHOUT TIMEZONE]
CHAR(n) VARCHAR(n) TEXT	CHAR(n) CHARACTER(n) VARCHAR(n) CHARACTER VARYING(n) TEXT	STRING
BINARY VARBINARY BLOB	BYTEA	BYTES
-	ARRAY	ARRAY

示例

从Kafka源表中读取数据，将JDBC表作为维表，并将二者生成的表信息写入Kafka结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，在DLI上根据MySQL和Kafka所在的虚拟私有云和子网分别创建相应的增强型跨源连接，并绑定所要使用的Flink弹性资源池。
2. 设置MySQL和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**分别根据MySQL和Kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 连接MySQL数据库实例，在flink数据库中创建相应的表，作为维表，表名为area_info，SQL语句如下：

```
CREATE TABLE `flink`.`area_info` (
    `area_id` VARCHAR(32) NOT NULL,
    `area_province_name` VARCHAR(32) NOT NULL,
    `area_city_name` VARCHAR(32) NOT NULL,
    `area_county_name` VARCHAR(32) NOT NULL,
    `area_street_name` VARCHAR(32) NOT NULL,
    `region_name` VARCHAR(32) NOT NULL,
    PRIMARY KEY (`area_id`)
)   ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4
    COLLATE = utf8mb4_general_ci;
```

4. 连接MySQL数据库实例，向JDBC维表area_info中插入测试数据，其语句如下：

```
insert into flink.area_info
  (area_id, area_province_name, area_city_name, area_county_name, area_street_name, region_name)
  values
  ('330102', 'a1', 'b1', 'c1', 'd1', 'e1'),
  ('330106', 'a1', 'b1', 'c2', 'd2', 'e1'),
  ('330108', 'a1', 'b1', 'c3', 'd3', 'e1'),  ('330110', 'a1', 'b1', 'c4', 'd4', 'e1');
```


版权所有 © 华为技术有限公司

5. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。该作业脚本将Kafka为数据源，JDBC作为维表，数据写入到Kafka结果表。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '***KafkaSourceTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = 'jdbc-order',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

--创建地址维表
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://***JDBCAddress:JDBCPort***/flink',--其中url中的flink表示MySQL中area_info表所在的数据库名
  'table-name' = 'area_info',
  'username' = '***JDBCUserName***',
  'password' = '***JDBCPassWord***'
);

--根据地址维表生成详细的包含地址的订单信息宽表
create table order_detail(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) with (
  'connector' = 'kafka',
  'topic' = '***KafkaSinkTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'format' = 'json'
);

insert into order_detail
```

 版权所有 © 华为技术有限公司

```
  select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
        area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
        area.area_street_name, area.region_name  from orders
        left join area_info for system_time as of orders.proctime as area on orders.area_id =
area.area_id;
```

6. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",
"user_name":"Cindy", "area_id":"330108"}
```

7. 连接Kafka集群，在Kafka的sink topic读取数据，结果参考如下：

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_ci
ty_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}

{"order_id":"202103251505050001","order_channel":"qqShop","order_time":"2021-03-25
15:05:05","pay_amount":500.0,"real_pay":400.0,"pay_time":"2021-03-25
15:10:00","user_id":"0003","user_name":"Cindy","area_id":"330108","area_province_name":"a1","area_c
ity_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

常见问题

无。

3.3.3.4 Redis 维表

功能描述

创建Redis表作为维表用于与输入流连接，从而生成相应的宽表。

前提条件

- 要建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。
- Flink跨源开发场景中直接配置跨源认证信息存在密码泄露的风险，优先推荐您使用DLI提供的跨源认证。

  跨源认证简介及操作方法请参考**跨源认证简介**。

注意事项

- 创建Flink OpenSource SQL作业时，在作业编辑界面的“运行参数”处，“Flink版本”需要选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。

- 若需要获取key的值，则可以通过在flink中设置主键获取，主键字段即对应redis的key。
- 若定义主键，则不能够定义复合主键，即主键只能是一个字段，不能是多个字段。
- schema-syntax取值约束：
  - 当schema-syntax为map或array时，非主键字段最多只能只有一个，且需要为相应的map或array类型。
  - 当schema-syntax为fields-scores时，非主键字段个数需要为偶数，且除主键字段外，每两个字段的第二个字段的类型需要为doule，会将该字段的值视为前一个字段的score，其示例如下：

```
CREATE TABLE redisSource (
  redisKey string,
  order_id string,
  score1 double,
  order_channel string,
  score2 double,
  order_time string,
  score3 double,
  pay_amount double,
  score4 double,
  real_pay double,
  score5 double,
  pay_time string,
  score6 double,
  user_id string,
  score7 double,
  user_name string,
  score8 double,
  area_id string,
  score9 double,
  primary key (redisKey) not enforced
) WITH (
  'connector' = 'redis',
  'host' = 'RedisIP',
  'password' = 'RedisPassword',
  'data-type' = 'sorted-set',
  'deploy-mode' = 'master-replica',
  'schema-syntax' = 'fields-scores'
);
```

- data-type取值约束：
  - 当data-type为set时，flink中定义的非主键字段的类型必须相同。
  - 当data-type为sorted-set且schema-syntax为fields和array时，只能读取redis的sorted set中的值，而不能读取score。
  - 当data-type为string时，只能有一个非主键字段。
  - 当data-type为sorted-set，且schema-syntax为map时，除主键字段外，只能有一个非主键字段，且需要为map类型，同时该字段的map的value需要为double类型，表示score，该字段的map的key表示redis的set中的值。
  - 当data-type为sorted-set，且schema-syntax为array-scores时，除主键字段外，只能有两个非主键字段，且这两个字段的类型需要为array。

    两个字段其中第一个字段类型是array表示Redis的set中的值，第二个字段类型为array<double>，表示相应索引的score。其示例如下：

```
CREATE TABLE redisSink (
  order_id string,
  arrayField Array<String>,
  arrayScore array<double>,
  primary key (order_id) not enforced
) WITH (
  'connector' = 'redis',
```

 版权所有 © 华为技术有限公司

```
 'host' = 'RedisIP',
 'password' = 'RedisPassword',
 'data-type' = 'sorted-set',
 "default-score" = '3',
 'deploy-mode' = 'master-replica',
 'schema-syntax' = 'array-scores'
);
```

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
  ,PRIMARY KEY (attr_name, ...) NOT ENFORCED
)
with (
  'connector' = 'redis',
  'host' = ''
);
```

参数说明

表 3-32 参数说明

参数	是否必选	默认值	数据类型	说明
connector	是	无	String	connector类型，需配置为'redis'。
host	是	无	String	redis连接地址。
port	否	6379	Integer	redis连接端口。
password	否	无	String	redis认证密码。
namespace	否	无	String	redis key的namespace
delimiter	否	:	String	redis的key和namespace之间的分隔符。
data-type	否	hash	String	redis的数据类型，有下列选项 • hash • list • set • sorted-set • string data-type取值约束详见**data-type取值约束**说明。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
schema-syntax	否	fields	String	redis的schema语义，包含以下值： • fields：适用于所有数据类型 • fields-scores：适用于sorted set数据类型 • array：适用于list、set、sorted set数据类型 • array-scores：适用于sorted set数据类型 • map：适用于hash、sorted set数据类型 schema-syntax取值约束详见**schema-syntax取值约束**说明。
deploy-mode	否	standalone	String	redis集群的部署模式，支持standalone、master-replica、cluster，默认standalone。
retry-count	是	5	Integer	设置每个连接请求的队列大小。如果超过队列大小，则命令调用将导致RedisException。将requestQueueSize设置为较低的值将导致在过载期间或连接处于断开状态时更早出现异常。更高的值意味着达到边界需要更长的时间，但可能会有更多的请求排队，并使用更多的堆空间。默认请设置为2147483647。
connection-timeout-millis	否	10000	Integer	尝试连接redis集群时的最大超时时间。
commands-timeout-millis	否	2000	Integer	等待操作完成响应的最大时间。
rebalancing-timeout-millis	否	15000	Integer	redis集群失败时的休眠时间。
scan-keys-count	否	1000	Integer	每次扫描时读取的数量。
default-score	否	0	Double	当data-type设置为“sorted-set”数据类型的默认score。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	数据类型	说明
deserialize-error-policy	否	fail-job	Enum	数据解析失败时的处理方式。 枚举类型，包含以下值： • fail-job：作业失败 • skip-row：跳过当前数据 • null-field：设置当前数据为null
skip-null-values	否	true	Boolean	是否跳过null。
lookup.async	否	false	Boolean	作为redis维表时，是否使用异步I/O。
pwd_auth_name	否	无	String	DLI侧创建的Password类型的跨源认证名称。 使用跨源认证则无需在作业中配置和账号密码。

示例

从Kafka源表中读取数据，将Redis表作为维表，并将二者生成的宽表信息写入Kafka结果表中，其具体步骤如下：

1. 参考**增强型跨源连接**，根据Redis和Kafka所在的虚拟私有云和子网创建相应的增强型跨源，并绑定所要使用的Flink弹性资源池。
2. 设置Redis和Kafka的安全组，添加入向规则使其对Flink的队列网段放通。参考**测试地址连通性**根据Redis的地址测试队列连通性。若能连通，则表示跨源已经绑定成功，否则表示未成功。
3. 登录Redis客户端，通过如下命令向Redis发送如下数据：

```
HMSET 330102  area_province_name a1 area_province_name b1 area_county_name c1 area_street_name d1 region_name e1

HMSET 330106  area_province_name a1 area_province_name b1 area_county_name c2 area_street_name d2 region_name e1

HMSET 330108  area_province_name a1 area_province_name b1 area_county_name c3 area_street_name d3 region_name e1

HMSET 330110  area_province_name a1 area_province_name b1 area_county_name c4 area_street_name d4 region_name e1
```

4. 参考**创建Flink OpenSource作业**，创建flink opensource sql作业，输入以下作业脚本，提交运行作业。该作业脚本将Kafka为数据源，Redis作为维表，数据写入到Kafka结果表中。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
```


版权所有 © 华为技术有限公司

```
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '***kafkaSourceTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

--创建地址维表
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string,
    primary key (area_id) not enforced -- redis的key
) WITH (
  'connector' = 'redis',
  'host' = '***RedisIP***',
  'password' = '***RedisPassword***',
  'data-type' = 'hash',
  'deploy-mode' = 'master-replica'
);

--根据地址维表生成详细的包含地址的订单信息宽表
create table order_detail(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) with (
  'connector' = 'kafka',
  'topic' = '***kafkaSinkTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'format' = 'json'
);

insert into order_detail
    select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
           area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
           area.area_street_name, area.region_name  from orders
    left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

5. 连接Kafka集群，向Kafka的source topic中插入如下测试数据：

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
```

 版权所有 © 华为技术有限公司

```
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}

{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",
"user_name":"Cindy", "area_id":"330108"}
```

6. 连接Kafka集群，在Kafka的sink topic读取数据，结果数据参考如下：

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106","area_province_name":"a1","area_ci
ty_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"202103251202020001","order_channel":"miniAppShop","order_time":"2021-03-25
12:02:02","pay_amount":60.0,"real_pay":60.0,"pay_time":"2021-03-25
12:03:00","user_id":"0002","user_name":"Bob","area_id":"330110","area_province_name":"a1","area_cit
y_name":"b1","area_county_name":"c4","area_street_name":"d4","region_name":"e1"}

{"order_id":"202103251505050001","order_channel":"qqShop","order_time":"2021-03-25
15:05:05","pay_amount":500.0,"real_pay":400.0,"pay_time":"2021-03-25
15:10:00","user_id":"0003","user_name":"Cindy","area_id":"330108","area_province_name":"a1","area_c
ity_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

常见问题

若在windows环境中向redis中写入中文时，会导致写入数据异常，请避免此情况。

3.3.4 Format

3.3.4.1 Avro Format

功能描述

Avro格式允许基于Avro schema 读取和写入Avro 数据。目前，Avro schema 从表 schema 推导。

支持的 Connector

- Kafka
- Upsert Kafka

参数说明

表 3-33 参数说明

参数	是否必选	默认值	类型	说明
format	是	（none）	String	指定使用格式，这里应该是 'avro'。
avro.codec	否	（none）	String	仅用于文件系统，avro 压缩编解码器。默认不压缩。目前支持：deflate、snappy、bzip2、xz。

 版权所有 © 华为技术有限公司

数据类型映射

目前，Avro schema 通常是从 table schema 中推导而来。尚不支持显式定义 Avro schema。因此，下表列出了从 Flink 类型到 Avro 类型的类型映射。

除了下面列出的类型，Flink 支持读取/写入 nullable 的类型。Flink 将 nullable 的类型映射到 Avro union(something, null)，其中 something 是从 Flink 类型转换的 Avro 类型。

表 3-34 数据类型映射

Flink SQL类型	Avro类型	Avro逻辑类型
CHAR / VARCHAR / STRING	string	-
BOOLEAN	boolean	-
BINARY / VARBINARY	bytes	-
DECIMAL	fixed	decimal
TINYINT	int	-
SMALLINT	int	-
INT	int	-
BIGINT	long	-
FLOAT	float	-
DOUBLE	double	-
DATE	int	date
TIME	int	time-millis
TIMESTAMP	long	timestamp-millis
ARRAY	array	-
MAP(key 必须是 string/char/varchar 类型)	map	-
MULTISET(元素必须是 string/char/varchar 类型)	map	-
ROW	record	-

示例

读取kafka中的数据，以avro格式反序列化，并输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列-->找到作业的所属队列-->更多-->测试地址连通性-->输入kafka的地址-->测试）。若能连通，则表示跨源已经绑定成功；否则表示未成功。

 版权所有 © 华为技术有限公司

步骤2 创建flink opensource sql作业，选择flink1.12，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>,<yourKafkaAddress3>:<yourK
afkaPort>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  "format" = "avro"
);

CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);

insert into printSink select * from kafkaSource;
```

步骤3 向kafka中以avro的序列化方式插入如下数据：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

步骤4 用户可按下述操作查看输出结果:

- 方法一："更多" -> "FlinkUI" -> "Task Managers" -> "Stdout"。
- 方法二：若在提交运行作业前选择了保存日志，则可以从日志的taskmanager.out文件中查看。

  ```
  +I(202103241000000001,webShop,2021-03-2410:00:00,100.0,100.0,2021-03-2410:02:03,0001,Alice,330
  106)
  +I(202103241606060001,appShop,2021-03-2416:06:06,200.0,180.0,2021-03-2416:10:06,0001,Alice,3301
  06)
  ```

----结束

 版权所有 © 华为技术有限公司

3.3.4.2 Canal Format

功能描述

Canal是一个 CDC（ChangeLog Data Capture，变更日志数据捕获）工具，可以实时地将 MySQL 变更传输到其他系统。Canal 为变更日志提供了统一的数据格式，并支持使用 JSON 或 protobuf序列化消息（Canal 默认使用 protobuf）。

Flink 支持将 Canal 的 JSON 消息解析为 INSERT / UPDATE / DELETE 消息到 Flink SQL 系统中。在很多情况下，利用这个特性非常的有用，例如

- 将增量数据从数据库同步到其他系统
- 日志审计
- 数据库的实时物化视图
- 关联维度数据库的变更历史，等等。

Flink 还支持将 Flink SQL 中的 INSERT / UPDATE / DELETE 消息编码为 Canal 格式的 JSON 消息，输出到 Kafka 等存储中。 但需要注意的是，目前 Flink 还不支持将 UPDATE_BEFORE 和 UPDATE_AFTER 合并为一条 UPDATE 消息。因此，Flink 将 UPDATE_BEFORE 和 UPDATE_AFTER 分别编码为 DELETE 和 INSERT 类型的 Canal 消息。

参数说明

表 3-35 参数说明

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定要使用的格式，此处应为 'canal-json'.
canal-json.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。
canal-json.timestamp-format.standard	否	'SQL'	String	指定输入和输出时间戳格式。当前支持的值是：'SQL'和'ISO-8601'。 ● 选项 'SQL' 将解析 "yyyy-MM-dd HH:mm:ss.s{precision}" 格式的输入时间戳，例如 '2020-12-30 12:13:14.123'，并以相同格式输出时间戳。 ● 选项 'ISO-8601' 将解析 "yyyy-MM-ddTHH:mm:ss.s{precision}" 格式的输入时间戳，例如 '2020-12-30T12:13:14.123'，并以相同的格式输出时间戳。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
canal-json.map-null-key.mode	否	'FALL'	String	指定处理 Map 中 key 值为空的方法. 当前支持的值有'FAIL', 'DROP'和'LITERAL'。 ● Option 'FAIL' 将抛出异常，如果遇到 Map 中 key 值为空的数据。 ● Option 'DROP' 将丢弃 Map 中 key 值为空的数据项。 ● Option 'LITERAL' 将使用字符串常量来替换 Map 中的空 key 值。字符串常量的值由 'canal-json.map-null-key.literal' 定义。
canal-json.map-null-key.literal	否	'null'	String	当 'canal-json.map-null-key.mode' 是 LITERAL 的时候，指定字符串常量替换 Map 中的空 key 值。
canal-json.database.include	否	(none)	String	仅读取指定数据库的 changelog 记录（通过对比 Canal 记录中的 "database" 元数据字段）。
canal-json.table.include	否	(none)	String	仅读取指定表的 changelog 记录（通过对比 Canal 记录中的 "table" 元数据字段）。

支持的 Connector

- Kafka

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列-->找到作业的所属队列-->更多-->测试地址连通性-->输入kafka的地址-->测试）。若能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.12版本，并提交运行，其代码如下：

```
create table kafkaSource(
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
  ) with (
    'connector' = 'kafka',
    'topic' = '<yourTopic>',
    'properties.group.id' = '<yourGroupId>',
    'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'canal-json'
);
```


版权所有 © 华为技术有限公司

```
create table printSink(
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
   ) with (
     'connector' = 'print'
   );
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据：

```
{
  "data": [
    {
      "id": "111",
      "name": "scooter",
      "description": "Big 2-wheel scooter",
      "weight": "5.18"
    }
  ],
  "database": "inventory",
  "es": 1589373560000,
  "id": 9,
  "isDdl": false,
  "mysqlType": {
    "id": "INTEGER",
    "name": "VARCHAR(255)",
    "description": "VARCHAR(512)",
    "weight": "FLOAT"
  },
  "old": [
    {
      "weight": "5.15"
    }
  ],
  "pkNames": [
    "id"
  ],
  "sql": "",
  "sqlType": {
    "id": 4,
    "name": 12,
    "description": 12,
    "weight": 7
  },
  "table": "products",
  "ts": 1589373560798,
  "type": "UPDATE"
}
```

步骤4 用户可按下述操作查看输出结果:

- 方法一："更多" -> "FlinkUI" -> "Task Managers" -> "Stdout"。
- 方法二：若在提交运行作业前选择了保存日志，则可以从日志的taskmanager.out文件中查看。

```
-U(111,scooter,Big2-wheel scooter,5.15)
+U(111,scooter,Big2-wheel scooter,5.18)
```

----结束

 版权所有 © 华为技术有限公司

3.3.4.3 Confluent Avro Format

功能描述

Avro Schema Registry (avro-confluent) 格式能让您读取被 io.confluent.kafka.serializers.KafkaAvroSerializer序列化的记录，以及可以写入成能被 io.confluent.kafka.serializers.KafkaAvroDeserializer反序列化的记录。

当以这种格式读取（反序列化）记录时，将根据记录中编码的 schema 版本 id 从配置的 Confluent Schema Registry 中获取 Avro writer schema ，而从 table schema 中推断出 reader schema。

当以这种格式写入（序列化）记录时，Avro schema 是从 table schema 中推断出来的，并会用来检索要与数据一起编码的 schema id。我们会在配置的 Confluent Schema Registry 中配置的 **subject** 下，检索 schema id。subject 通过 avro-confluent.schema-registry.subject 参数来指定。

支持的 connector

- kafka
- upsert kafka

参数说明

表 3-36 参数说明

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定使用格式，这里应该是'avro-confluent'。
avro-confluent.schema-registry.subject	否	(none)	String	序列化期间，Confluent Schema Registry中注册schema所在的subject。 对于kafka和upsert-kafka，默认subject值是'<topic_name>-value' 或 '<topic_name>-key'
avro-confluent.schema-registry.url	是	(none)	String	注册或抓取schema的Confluent Schema Registry的URL。

示例

1. 从kafka中作为source的topic中读取json数据，并以confluent avro的形式写入作为sink的topic中

步骤1 根据kafka和ecs所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka和ecs的地址测试队列连通性（通用队列-->找到作业的所属队列-->更多-->测试地址连通性-->输入kafka或ecs的地址-->测试）。若能连通，则表示跨源已经绑定成功；否则表示未成功。

版权所有 © 华为技术有限公司

步骤2 购买ecs集群，并下载5.5.2版本的confluent（**https://packages.confluent.io/archive/5.5/**）和jdk1.8.0_232，并上传到购买的ecs集群中，然后使用下述命令解压（假设解压目录分别为confluent-5.5.2和jdk1.8.0_232）。

```
tar zxvf confluent-5.5.2-2.11.tar.gz
tar zxvf jdk1.8.0_232.tar.gz
```

步骤3 使用下述命令在当前ecs集群中安装jdk1.8.0_232(其中<yourJdkPath>可以在jdk1.8.0_232文件夹下使用"pwd"查看)：

```
export JAVA_HOME=<yourJdkPath>
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
```

步骤4 进入confluent-5.5.2/etc/schema-registry/目录下，修改schema-registry.properties文件中如下配置项：

```
listeners=http://<yourEcsIp>:8081
kafkastore.bootstrap.servers=<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>
```

步骤5 将ecs切换到confluent-5.5.2目录下，使用下述命令启动confluent：

```
bin/schema-registry-start etc/schema-registry/schema-registry.properties
```

步骤6 创建flink opensource sql作业，选择版本flink 1.12，并选择保存日志，然后提交运行：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
  'topic' = '<yourSourceTopic>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
CREATE TABLE kafkaSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
  'topic' = '<yourSinkTopic>',
  'format' = 'avro-confluent',
  'avro-confluent.schema-registry.url' = 'http://<yourEcsIp>:8081',
  'avro-confluent.schema-registry.subject' = '<yourSubject>'
);
insert into kafkaSink select * from kafkaSource;
```

步骤7 向kafka中插入如下数据：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

 版权所有 © 华为技术有限公司

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

步骤8 读取kafka的作为sink的topic的数据，则可发现数据已经写入，且schema已经保存到kafka的_schema的topic中。

----结束

3.3.4.4 CSV Format

功能描述

CSV Format 允许我们基于CSV schema 进行解析和生成CSV 数据。目前的CSV schema 是基于table schema 推导出来的。

支持的 Connector

- Kafka
- Upsert Kafka

参数说明

表 3-37

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定要使用的格式，这里应该是 'csv'。
csv.field-delimiter	否	,	String	字段分隔符 (默认',')，必须为单字符。 您可以使用反斜杠字符指定一些特殊字符，例如 '\t' 代表制表符。 您也可以通过 unicode 编码在纯 SQL 文本中指定一些特殊字符，例如 'csv.field-delimiter' = '\u0001' 代表 0x01 字符。
csv.disable-quote-character	否	false	Boolean	是否禁止对引用的值使用引号 (默认是 false). 如果禁止，选项 'csv.quote-character' 不能设置。
csv.quote-character	否	‘’	String	用于围住字段值的引号字符 (默认").
csv.allow-comments	否	false	Boolean	是否允许忽略注释行（默认不允许），注释行以 '#' 作为起始字符。 如果允许注释行，请确保 csv.ignore-parse-errors 也开启了从而允许空行。
csv.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
csv.array-element-delimiter	否	;	String	分隔数组和行元素的字符串(默认';').
csv.escape-character	否	(none)	String	转义字符(默认关闭).
csv.null-literal	否	(none)	String	是否将 "null" 字符串转化为 null 值。

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列-->找到作业的所属队列-->更多-->测试地址连通性-->输入kafka的地址-->测试）。若能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，并提交运行，其代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSourceTopic>',
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  "format" = "csv"
);

CREATE TABLE kafkaSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSinkTopic>',
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
  "format" = "csv"
);

insert into kafkaSink select * from kafkaSource;
```


版权所有 © 华为技术有限公司

步骤3 向kafka的作为source的topic中插入下列数据：

```
20210325150505001,qqShop,2021-03-25 15:05:05,500.00,400.00,2021-03-25 15:10:00,0003,Cindy,330108

20210324160606001,appShop,2021-03-24 16:06:06,200.00,180.00,2021-03-24 16:10:06,0001,Alice,330106
```

步骤4 读取kafka中作为sink的topic，结果如下：

```
20210325150505001,qqShop,"2021-03-25 15:05:05",500.0,400.0,"2021-03-25 15:10:00",0003,Cindy,330108

20210324160606001,appShop,"2021-03-24 16:06:06",200.0,180.0,"2021-03-24 16:10:06",0001,Alice,330106
```

----结束

3.3.4.5 Debezium Format

功能描述

Debezium是一个 CDC（Changelog Data Capture，变更数据捕获）的工具，可以把其他数据库的更改实时流式传输到 Kafka 中。 Debezium 为变更日志提供了统一的格式结构，并支持使用 JSON消息。

Flink 支持将 Debezium JSON解析为 INSERT / UPDATE / DELETE 消息到 Flink SQL 系统中。在很多情况下，利用这个特性非常的有用，例如

- 将增量数据从数据库同步到其他系统
- 日志审计
- 数据库的实时物化视图
- 关联维度数据库的变更历史，等等。

参数说明

表 3-38

参数	是否必选	默认值	是否必选	描述
format	是	(none)	String	指定要使用的格式，此处应为'debezium-json'。
debezium-json.schema-include	否	false	Boolean	设置 Debezium Kafka Connect 时，用户可以启用 Kafka 配置'value.converter.schemas.enable' 以在消息中包含 schema。此选项表明 Debezium JSON 消息是否包含 schema。
debezium-json.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。

 版权所有 © 华为技术有限公司

参数	是否必选	默认值	是否必选	描述
debezium-json.timestamp-format.standard	否	'SQL'	String	声明输入和输出的时间戳格式。当前支持的格式为'SQL'和'ISO-8601'。 ● 可选参数 'SQL' 将会以 "yyyy-MM-dd HH:mm:ss.s{precision}" 的格式解析时间戳, 例如 '2020-12-30 12:13:14.123'，且会以相同的格式输出。 ● 可选参数 'ISO-8601' 将会以 "yyyy-MM-ddTHH:mm:ss.s{precision}" 的格式解析输入时间戳, 例如 '2020-12-30T12:13:14.123' ，且会以相同的格式输出。
debezium-json.map-null-key.mode	否	'FAIL'	String	指定处理 Map 中 key 值为空的方法。当前支持的值有FAIL、DROP和LITERAL。 ● Option 'FAIL' 将抛出异常，如果遇到 Map 中 key 值为空的数据。 ● Option 'DROP' 将丢弃 Map 中 key 值为空的数据项。 ● Option 'LITERAL' 将使用字符串常量来替换 Map 中的空 key 值。字符串常量的值由 'debezium-json.map-null-key.literal' 定义。
debezium-json.map-null-key.literal	否	'null'	String	当 'debezium-json.map-null-key.mode' 是 LITERAL 的时候，指定字符串常量替换 Map 中的空 key 值。

支持的 Connector

- Kafka

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列-->找到作业的所属队列-->更多-->测试地址连通性-->输入kafka的地址-->测试）。若能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，并提交运行，其代码如下：

```
create table kafkaSource(
  id BIGINT,
  name STRING,
  description STRING,
  weight DECIMAL(10, 2)
```

 版权所有 © 华为技术有限公司

```
 ) with (
   'connector' = 'kafka',
   'topic' = '<yourTopic>',
   'properties.group.id' = '<yourGroupId>',
   'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
   'scan.startup.mode' = 'latest-offset',
   'format' = 'debezium-json'
);
create table printSink(
  id BIGINT,
  name STRING,
  description STRING,
  weight DECIMAL(10, 2)
   ) with (
     'connector' = 'print'
   );
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据：

```
{
  "before": {
    "id": 111,
    "name": "scooter",
    "description": "Big 2-wheel scooter",
    "weight": 5.18
  },
  "after": {
    "id": 111,
    "name": "scooter",
    "description": "Big 2-wheel scooter",
    "weight": 5.15
  },
  "source": {
    "version": "0.9.5.Final",
    "connector": "mysql",
    "name": "fullfillment",
    "server_id" :1,
    "ts_sec": 1629607909,
    "gtid": "mysql-bin.000001",
    "pos": 2238,"row": 0,
    "snapshot": false,
    "thread": 7,
    "db": "inventory",
    "table": "test",
    "query": null},
  "op": "u",
  "ts_ms": 1589362330904,
  "transaction": null
}
```

步骤4 用户可按下述操作查看输出结果:

- 方法一：“更多” -> "FlinkUI" -> "Task Managers" -> "Stdout"。
- 方法二：若在提交运行作业前选择了保存日志，则可以从日志的taskmanager.out文件中查看。

```
-U(111,scooter,Big2-wheel scooter,5.18)
+U(111,scooter,Big2-wheel scooter,5.15)
```

----结束

3.3.4.6 JSON Format

功能描述

JSON Format 能读写 JSON 格式的数据。当前，JSON schema 是从 table schema 中自动推导而得的。

支持的 Connector

- Kafka
- Upsert Kafka
- Elasticsearch

参数说明

表 3-39

参数	是否必选	默认值	类型	说明
format	是	(none)	String	声明使用的格式，这里应为'json'。
json.fail-on-missing-field	否	false	Boolean	当解析字段缺失时，是跳过当前字段或行，还是抛出错误失败（默认为 false，不抛出错误失败）。
json.ignore-parse-errors	否	false	Boolean	当解析异常时，是跳过当前字段或行，还是抛出错误失败（默认为 false，即抛出错误失败）。如果忽略字段的解析异常，则会将该字段值设置为null。

版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	说明
json.timestamp-format.standard	否	'SQL'	String	声明输入和输出的TIMESTAMP和TIMESTAMP WITH LOCAL TIME ZONE 的格式。 当前支持的格式为'SQL'和'ISO-8601'： ● 可选参数 'SQL' 将会以 "yyyy-MM-dd HH:mm:ss.s{precision}" 的格式解析 TIMESTAMP, 例如 "2020-12-30 12:13:14.123"，以 "yyyy-MM-dd HH:mm:ss.s{precision}'Z'" 的格式解析 TIMESTAMP WITH LOCAL TIME ZONE, 例如 "2020-12-30 12:13:14.123Z" 且会以相同的格式输出。 ● 可选参数 'ISO-8601' 将会以 "yyyy-MM-ddTHH:mm:ss.s{precision}" 的格式解析输入 TIMESTAMP, 例如 "2020-12-30T12:13:14.123" ，以 "yyyy-MM-ddTHH:mm:ss.s{precision}'Z'" 的格式解析 TIMESTAMP WITH LOCAL TIME ZONE, 例如 "2020-12-30T12:13:14.123Z" 且会以相同的格式输出。
json.map-null-key.mode	否	'FALL'	String	指定处理 Map 中 key 值为空的方法。当前支持的值有：'FAIL'，'DROP'和'LITERAL'。 ● Option 'FAIL' 将抛出异常，如果遇到 Map 中 key 值为空的数据。 ● Option 'DROP' 将丢弃 Map 中 key 值为空的数据项。 ● Option 'LITERAL' 将使用字符串常量来替换 Map 中的空 key 值。字符串常量的值由 'json.map-null-key.literal' 定义。
json.map-null-key.literal	否	'null'	String	当 'json.map-null-key.mode' 是LITERAL的时候，指定字符串常量替换 Map 中的空 key 值。

版权所有 © 华为技术有限公司

示例

该示例是从kafka的一个topic中读取数据，并使用kafka sink将数据写入到kafka的另一个topic中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功；否则表示未成功

步骤2 创建flink opensource sql作业，并选择flink版本为1.12，选择保存日志，然后提交并运行，其SQL代码如下：

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSourceTopic>',
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE kafkaSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSinkTopic>',
  'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
  "format" = "json"
);

insert into kafkaSink select * from kafkaSource;
```

步骤3 向作为source的kafka的topic中插入下列数据：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}

{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

步骤4 读取作为sink的kafka的topic中的数据，其结果如下：

```
{"order_id":"202103241000000001","order_channel":"webShop","order_time":"2021-03-24
10:00:00","pay_amount":100.0,"real_pay":100.0,"pay_time":"2021-03-24
10:02:03","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

 版权所有 © 华为技术有限公司

```
{"order_id":"202103241606060001","order_channel":"appShop","order_time":"2021-03-24
16:06:06","pay_amount":200.0,"real_pay":180.0,"pay_time":"2021-03-24
16:10:06","user_id":"0001","user_name":"Alice","area_id":"330106"}
```

----结束

3.3.4.7 Maxwell Format

功能描述

Flink 支持将 Maxwell JSON 消息解释为 INSERT/UPDATE/DELETE 消息到 Flink SQL 系统中。在许多情况下，这对于利用此功能很有用。

例如：

- 将数据库中的增量数据同步到其他系统
- 审计日志
- 数据库的实时物化视图
- 临时连接更改数据库表的历史等等。

Flink 还支持将 Flink SQL 中的 INSERT/UPDATE/DELETE 消息编码为 Maxwell JSON 消息，并发送到 Kafka 等外部系统。但是，目前 Flink 无法将 UPDATE_BEFORE 和 UPDATE_AFTER 合并为一条 UPDATE 消息。因此，Flink 将 UPDATE_BEFORE 和 UDPATE_AFTER 编码为 DELETE 和 INSERT Maxwell 消息。

参数说明

参数	是否必选	默认值	类型	说明
format	是	(none)	String	指定使用格式，此处使用'maxwell-json'。
maxwell-json.ignore-parse-errors	否	false	Boolean	跳过解析错误而不是失败的字段和行。出现错误时，字段设置为空。
maxwell-json.timestamp-format.standard	否	'SQL'	String	指定输入和输出时间戳格式。当前支持的值为“SQL”和“ISO-8601”：选项“SQL”将以“yyyy-MM-dd HH:mm:ss.s{precision}”格式解析输入时间戳，例如“2020-12-30 12” :13:14.123' 并以相同格式输出时间戳。选项'ISO-8601' 将以“yyyy-MM-ddTHH:mm:ss.s{precision}”格式解析输入时间戳，例如'2020-12-30T12：13:14.123' 并以相同格式输出时间戳。

参数	是否必选	默认值	类型	说明
maxwell-json.map-null-key.mode	否	'FAIL'	String	在序列化地图数据的空键时指定处理模式。当前支持的值为“FAIL”、“DROP”和“LITERAL”：选项“FAIL”将在遇到带有空键的地图时抛出异常。选项“DROP”将删除地图数据的空键条目。选项“LITERAL”将替换空带字符串文字的键。字符串文字由 maxwell-json.map-null-key.literal 选项定义。
maxwell-json.map-null-key.literal	否	'null'	String	当 'maxwell-json.map-null-key.mode' 为 LITERAL 时，指定字符串文字以替换空键。

支持的 Connector

- Kafka

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列-->找到作业的所属队列-->更多-->测试地址连通性-->输入kafka的地址-->测试）。若能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.12，并提交运行，其代码如下：

```
create table kafkaSource(
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
  ) with (
    'connector' = 'kafka',
    'topic' = '<yourTopic>',
    'properties.group.id' = '<yourGroupId>',
    'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'maxwell-json'
);
create table printSink(
  id bigint,
  name string,
  description string,
  weight DECIMAL(10, 2)
   ) with (
     'connector' = 'print'
   );
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据：

```
{
  "database":"test",
```


版权所有 © 华为技术有限公司

```
  "table":"e",
  "type":"insert",
  "ts":1477053217,
  "xid":23396,
  "commit":true,
  "position":"master.000006:800911",
  "server_id":23042,
  "thread_id":108,
  "primary_key": [1, "2016-10-21 05:33:37.523000"],
  "primary_key_columns": ["id", "c"],
  "data":{
    "id":111,
    "name":"scooter",
    "description":"Big 2-wheel scooter",
    "weight":5.15
  },
  "old":{
    "weight":5.18
  }
}
```

步骤4 用户可按下述操作查看输出结果:

- 方法一："更多" -> "FlinkUI" -> "Task Managers" -> "Stdout"。
- 方法二：若在提交运行作业前选择了保存日志，则可以从日志的taskmanager.out文件中查看。

```
+I(111,scooter,Big 2-wheel scooter,5.15)
```

----结束

3.3.4.8 Raw Format

功能描述

Raw format 允许读写原始（基于字节）值作为单个列。

注意: 这种格式将 null 值编码成 byte[] 类型的 null。这样在 upsert-kafka 中使用时可能会有限制，因为 upsert-kafka 将 null 值视为 墓碑消息（在键上删除）。因此，如果该字段可能具有 null 值，我们建议避免使用 upsert-kafka 连接器和 raw format 作为 value.format。

Raw format 连接器是内置的。

参数说明

表 3-40

参数	是否必选	默认值	类型	描述
format	是	(none)	String	指定要使用的格式, 这里应该是 'raw'。
raw.charset	否	UTF-8	String	指定字符集来编码文本字符串。

 版权所有 © 华为技术有限公司

参数	是否必选	默认值	类型	描述
raw.endiannes s	否	big-endian	String	指定字节序来编码数字值的字节。有效值为'big-endian'和'little-endian'。

支持的 Connector

- Kafka
- UpsertKafka

示例

使用kafka发送数据，输出到print中。

步骤1 根据kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据kafka的地址测试队列连通性（通用队列-->找到作业的所属队列-->更多-->测试地址连通性-->输入kafka的地址-->测试）。若能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 创建flink opensource sql作业，选择flink1.12，并提交运行，其代码如下：

```
create table kafkaSource(
  log string
  ) with (
    'connector' = 'kafka',
    'topic' = '<yourTopic>',
    'properties.group.id' = '<yourGroupId>',
    'properties.bootstrap.servers' = '<yourKafkaAddress>:<yourKafkaPort>',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'raw'
);
create table printSink(
  log string
   ) with (
     'connector' = 'print'
   );
insert into printSink select * from kafkaSource;
```

步骤3 向kafka的相应topic中插入下列数据：

```
47.29.201.179 - - [28/Feb/2019:13:17:10 +0000] "GET /?p=1 HTTP/2.0" 200 5316 "https://domain.com/?p=1" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36" "2.75"
```

步骤4 用户可按下述操作查看输出结果:

- 方法一："更多" -> "FlinkUI" -> "Task Managers" -> "Stdout"。
- 方法二：若在提交运行作业前选择了保存日志，则可以从日志的taskmanager.out文件中查看。

```
+I(47.29.201.179 - - [28/Feb/2019:13:17:10 +0000] "GET /?p=1 HTTP/2.0"2005316"https://domain.com/?p=1"
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36" "2.75")
```

----结束

 版权所有 © 华为技术有限公司

3.4 数据操作语句 DML

3.4.1 SELECT

SELECT

语法格式

```
SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

SELECT语句用于从表中选取数据。

ALL表示返回所有结果。

DISTINCT表示返回不重复结果。

注意事项

- 所查询的表必须是已经存在的表，否则会出错。
- WHERE关键字指定查询的过滤条件，过滤条件中支持算术运算符，关系运算符，逻辑运算符。
- GROUP BY指定分组的字段，可以单字段分组，也可以多字段分组。

示例

找出数量超过3的订单。

```
insert into temp SELECT  * FROM Orders WHERE units > 3;
```

插入一组常量数据。

```
insert into temp select 'Lily’, ‘male’, ‘student’, 17;
```

WHERE 过滤子句

语法格式

```
SELECT  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
```

语法说明

利用WHERE子句过滤查询结果。

注意事项

- 所查询的表必须是已经存在的，否则会出错。
- WHERE条件过滤，将不满足条件的记录过滤掉，返回满足要求的记录。

示例

 版权所有 © 华为技术有限公司

找出数量超过3并且小于10的订单。

```
insert into temp SELECT  * FROM Orders
  WHERE units > 3 and units < 10;
```

HAVING 过滤子句

功能描述

利用HAVING子句过滤查询结果。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤，HAVING子句支持算术运算，聚合函数等。

注意事项

如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。

示例

根据字段name对表student进行分组，再按组将score最大值大于95的记录筛选出来。

```
insert into temp SELECT name, max(score) FROM student
  GROUP BY name
  HAVING max(score) >95;
```

按列 GROUP BY

功能描述

按列进行分组操作。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
```

语法说明

GROUP BY：按列可分为单列GROUP BY与多列GROUP BY。

- 单列GROUP BY：指GROUP BY子句中仅包含一列。
- 多列GROUP BY：指GROUP BY子句中不止一列，查询语句将按照GROUP BY的所有字段分组，所有字段都相同的记录将被放在同一组中。

注意事项

GroupBy在流处理表中会产生更新结果

示例

 版权所有 © 华为技术有限公司

根据score及name两个字段对表student进行分组，并返回分组结果。

```
insert into temp SELECT name,score, max(score) FROM student
  GROUP BY name,score;
```

表达式 GROUP BY

功能描述

按表达式对流进行分组操作。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
```

语法说明

groupItem：可以是单字段，多字段，也可以是字符串函数等调用，不能是聚合函数。

注意事项

无

示例

先利用substring函数取字段name的子字符串，并按照该子字符串进行分组，返回每个子字符串及对应的记录数。

```
insert into temp SELECT substring(name,6),count(name) FROM student
  GROUP BY substring(name,6);
```

Grouping sets, Rollup, Cube

功能描述

- GROUPING SETS 的 GROUP BY 子句可以生成一个等效于由多个简单 GROUP BY 子句的 UNION ALL 生成的结果集，并且其效率比 GROUP BY 要高。
- ROLLUP与CUBE按一定的规则产生多种分组，然后按各种分组统计数据。
- CUBE生成的结果集显示了所选列中值的所有组合的聚合。
- Rollup生成的结果集显示了所选列中值的某一层次结构的聚合。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY groupingItem]
```

语法说明

groupingItem：是Grouping sets(columnName [, columnName]*)、Rollup(columnName [, columnName]*)、Cube(columnName [, columnName]*)

注意事项

无

示例

分别产生基于user和product的结果

 版权所有 © 华为技术有限公司

```
INSERT INTO temp SELECT SUM(amount)
FROM Orders
GROUP BY GROUPING SETS ((user), (product));
```

GROUP BY 中使用 HAVING 过滤

功能描述

利用HAVING子句在表分组后实现过滤。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。

注意事项

- 如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。HAVING与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。
- HAVING中除聚合函数外所使用的字段必须是GROUP BY中出现的字段。
- HAVING子句支持算术运算，聚合函数等。

示例

先依据num对表transactions进行分组，再利用HAVING子句对查询结果进行过滤，price与amount乘积的最大值大于5000的记录将被筛选出来，返回对应的num及price与amount乘积的最大值。

```
insert into temp SELECT num, max(price*amount) FROM transactions
  WHERE time > '2016-06-01'
  GROUP BY num
  HAVING max(price*amount)>5000;
```

3.4.2 集合操作

Union/Union ALL/Intersect/Except

语法格式

```
query UNION [ ALL ] | Intersect | Except query
```

语法说明

- UNION返回多个查询结果的并集。
- Intersect返回多个查询结果的交集。
- Except返回多个查询结果的差集。

注意事项

- 集合运算是以一定条件将表首尾相接，所以其中每一个SELECT语句返回的列数必须相同，列的类型一定要相同，列名不一定要相同。

 版权所有 © 华为技术有限公司

- UNION默认是去重的，UNION ALL是不去重的。

示例

输出Orders1和Orders2的并集，不包含重复记录。

```
insert into temp SELECT  * FROM Orders1
  UNION SELECT  * FROM Orders2;
```

IN

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  WHERE column_name IN (value (, value)* ) | query
```

语法说明

IN操作符允许在where子句中规定多个值。若表达式在给定的表子查询中存在，则返回 true 。

注意事项

子查询表必须由单个列构成，且该列的数据类型需与表达式保持一致。

示例

输出Orders中NewProducts中product的user和amount信息。

```
insert into temp SELECT user, amount
FROM Orders
WHERE product IN (
    SELECT product FROM NewProducts
);
```

3.4.3 窗口

GROUP WINDOW

语法说明

Group Window定义在GROUP BY里，每个分组只输出一条记录，包括以下几种：

- 分组函数

表 3-41 分组函数表

分组窗口函数	说明
TUMBLE(time_attr, interval)	定义一个滚动窗口。滚动窗口把行分配到有固定持续时间（ interval ）的不重叠的连续窗口。比如，5 分钟的滚动窗口以 5 分钟为间隔对行进行分组。滚动窗口可以定义在事件时间（批处理、流处理）或处理时间（流处理）上。

 版权所有 © 华为技术有限公司

分组窗口函数	说明
HOP(time_attr, interval, interval)	定义一个跳跃的时间窗口（在 Table API 中称为滑动窗口）。滑动窗口有一个固定的持续时间（ 第二个 interval 参数 ）以及一个滑动的间隔（第一个 interval 参数 ）。若滑动间隔小于窗口的持续时间，滑动窗口则会出现重叠；因此，行将会被分配到多个窗口中。比如，一个大小为 15 分钟的滑动窗口，其滑动间隔为 5 分钟，将会把每一行数据分配到 3 个 15 分钟的窗口中。滑动窗口可以定义在事件时间（批处理、流处理）或处理时间（流处理）上。
SESSION(time_attr, interval)	定义一个会话时间窗口。会话时间窗口没有一个固定的持续时间，但是它们的边界会根据 interval 所定义的不活跃时间所确定；即一个会话时间窗口在定义的间隔时间内没有事件出现，该窗口会被关闭。例如时间窗口的间隔时间是 30 分钟，当其不活跃的时间达到30分钟后，若观测到新的记录，则会启动一个新的会话时间窗口（否则该行数据会被添加到当前的窗口），且若在 30 分钟内没有观测到新记录，这个窗口将会被关闭。会话时间窗口可以使用事件时间（批处理、流处理）或处理时间（流处理）。

⚠ 注意

在流处理表中的 SQL 查询中，分组窗口函数的 time_attr 参数必须引用一个合法的时间属性，且该属性需要指定行的处理时间或事件时间。

- time_attr设置为event-time时参数类型为timestamp(3)类型。
- time_attr设置为processing-time时无需指定类型。

对于批处理的 SQL 查询，分组窗口函数的 time_attr 参数必须是一个timestamp类型的属性。

- 窗口辅助函数

  可以使用以下辅助函数选择组窗口的开始和结束时间戳以及时间属性

表 3-42 窗口辅助函数表

辅助函数	说明
TUMBLE_START(time_attr, interval) HOP_START(time_attr, interval, interval) SESSION_START(time_attr, interval)	返回相对应的滚动、滑动和会话窗口范围内的下界时间戳。

版权所有 © 华为技术有限公司

辅助函数	说明
TUMBLE_END(time_attr, interval) HOP_END(time_attr, interval, interval) SESSION_END(time_attr, interval)	返回相对应的滚动、滑动和会话窗口*范围以外*的上界时间戳。 注意： 范围以外的上界时间戳*不可以*在随后基于时间的操作中，作为行时间属性使用，比如基于时间窗口的join以及分组窗口或分组窗口上的聚合。
TUMBLE_ROWTIME(time_attr, interval) HOP_ROWTIME(time_attr, interval, interval) SESSION_ROWTIME(time_attr, interval)	返回的是一个可用于后续需要基于时间的操作的时间属性（rowtime attribute），比如基于时间窗口的join以及 分组窗口或分组窗口上的聚合。
TUMBLE_PROCTIME(time_attr, interval) HOP_PROCTIME(time_attr, interval, interval) SESSION_PROCTIME(time_attr, interval)	返回一个可用于后续需要基于时间的操作的 处理时间参数，比如基于时间窗口的join以及分组窗口或分组窗口上的聚合.

注意：辅助函数必须使用与GROUP BY 子句中的分组窗口函数完全相同的参数来调用.

示例

```
// 每天计算SUM（金额）（事件时间）。
insert into temp SELECT name,
    TUMBLE_START(ts, INTERVAL '1' DAY) as wStart,
    SUM(amount)
    FROM Orders
    GROUP BY TUMBLE(ts, INTERVAL '1' DAY), name;

// 每天计算SUM（金额）（处理时间）。
insert into temp SELECT name,
    SUM(amount)
    FROM Orders
    GROUP BY TUMBLE(proctime, INTERVAL '1' DAY), name;

// 每个小时计算事件时间中最近24小时的SUM（数量）。
insert into temp SELECT product,
    SUM(amount)
    FROM Orders
    GROUP BY HOP(ts, INTERVAL '1' HOUR, INTERVAL '1' DAY), product;

// 计算每个会话的SUM（数量），间隔12小时的不活动间隙（事件时间）。
insert into temp SELECT name,
    SESSION_START(ts, INTERVAL '12' HOUR) AS sStart,
    SESSION_END(ts, INTERVAL '12' HOUR) AS sEnd,
    SUM(amount)
    FROM Orders
    GROUP BY SESSION(ts, INTERVAL '12' HOUR), name;
```


版权所有 © 华为技术有限公司

TUMBLE WINDOW 扩展

功能描述

DLI TUMBLE函数功能增强主要包括以下功能：

- TUMBLE窗口周期性触发，控制延迟

  TUMBLE窗口结束之前，可以根据设置的触发频率周期性地触发窗口，输出从窗口开始时间到当前周期时间窗口内的计算结果值，但不影响最终窗口输出值，从而在窗口结束前的每个周期都可以看到最新的结果。

- 提高数据的精确性

  在窗口结束后，允许设置延迟时间。根据设置的延迟时间，每到达一个迟到数据，则更新窗口的输出结果

注意事项

- 若使用insert语句将结果写入sink中，则sink需要支持upsert模式，所以结果表需要支持upsert操作，且定义主键。
- 延迟时间设置仅用于事件时间，在处理时间中不生效。
- 辅助函数必须使用与 GROUP BY 子句中的分组窗口函数完全相同的参数来调用。
- 若使用事件时间，则需要使用watermark标识，代码如下（其中order_time被标识为事件时间列，watermark时间设置为3秒）：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  watermark for order_time as order_time - INTERVAL '3' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' = '<yourKafka>:<port>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
```

- 若使用处理时间，则需要使用计算列设置，其代码如下（其中proc即为处理时间列）：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proc as proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' = '<yourKafka>:<port>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);
```

 版权所有 © 华为技术有限公司

语法格式

```
TUMBLE(time_attr, window_interval, period_interval, lateness_interval)
```

语法示例

例如当前time_attr属性列为：testtime，窗口时间间隔为10秒，设置延迟时间为10秒
语法示例为：

```
TUMBLE(testtime, INTERVAL '10' SECOND, INTERVAL '10' SECOND, INTERVAL '10' SECOND)
```

参数说明

表 3-43 参数说明

参数	说明	参数格式
time_attr	表示相应的事件时间或者处理时间属性列。 ● time_attr设置为event-time时参数类型为timestamp(3)类型。 ● time_attr设置为processing-time时无需指定类型。	-
window_interval	表示窗口的持续时长。	● 格式1：**INTERVAL** '10' **SECOND** 表示窗口时间间隔为10秒，请根据实际情况修改该时间值。 ● 格式2：**INTERVAL** '10' **MINUTE** 表示窗口时间间隔为10分钟，请根据实际情况修改该时间值。 ● 格式3：**INTERVAL** '10' **DAY** 表示窗口时间间隔为10天，请根据实际情况修改该时间值。
period_interval	表示在窗口范围内周期性触发的频率，即在窗口结束前，从窗口开启开始，每隔period_interval时长更新一次输出结果。若没有设置，则默认没有使用周期触发策略。	
lateness_interval	表示窗口结束后延迟lateness_interval时长，继续统计在窗口结束后延迟时间内到达的属于该窗口的数据，而且在延迟时间内到达的每个数据都会更新输出结果。 **说明** 当时间窗口为处理时间时，无论lateness_interval为何值，都不会有效果。	

说明

period_interval和lateness_interval不可为负数。

- 当period_interval为0时，表示没有使用窗口的周期触发策略；
- 当lateness_interval为0时，表示没有使用窗口结束后的延迟策略；
- 当二者都没有填写时，默认两种策略都没有配置，仅使用普通的TUMBLE窗口。
- 若仅需使用延迟时间策略，则需要将上述period_interval格式中的'10'设置为 '0'。

辅助函数

 版权所有 © 华为技术有限公司

表 3-44 辅助函数

辅助函数	说明
TUMBLE_START(time_attr, window_interval, period_interval, lateness_interval)	返回相对应的滚动窗口范围内的下界时间戳。
TUMBLE_END(time_attr, window_interval, period_interval, lateness_interval)	返回相对应的滚动窗口范围以外的上界时间戳。

示例

1. 根据订单信息使用kafka作为数据源表，JDBC作为数据结果表统计用户在30秒内的订单数量，并根据窗口的订单id和窗口开启时间作为主键，将结果实时统计到JDBC中：

步骤1 根据MySQL和kafka所在的虚拟私有云和子网创建相应的跨源，并绑定所要使用的队列。然后设置安全组，入向规则，使其对当前将要使用的队列放开，并根据MySQL和kafka的地址测试队列连通性。若能连通，则表示跨源已经绑定成功；否则表示未成功。

步骤2 在MySQL的flink数据库下创建表order_count，创建语句如下：

```
CREATE TABLE `flink`.`order_count` (
    `user_id` VARCHAR(32) NOT NULL,
    `window_start` TIMESTAMP NOT NULL,
    `window_end` TIMESTAMP NULL,
    `total_num` BIGINT UNSIGNED NULL,
    PRIMARY KEY (`user_id`, `window_start`)
)   ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4
    COLLATE = utf8mb4_general_ci;
```

步骤3 创建flink opensource sql作业，并提交运行作业（这里设置窗口的大小为30秒，触发周期为10秒，延迟时间设置为5秒，即窗口结束前若结果有更新，则每隔十秒输出一次中间结果。在watermark到达使得窗口结束后，事件时间在watermark5秒内的数据仍然会被处理，并统计到当前所属窗口；若在5秒以外，则该数据会被丢弃）：

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time timestamp(3),
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  watermark for order_time as order_time - INTERVAL '3' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourTopic>',
  'properties.bootstrap.servers' = '<yourKafka>:<port>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE jdbcSink (
  user_id string,
  window_start timestamp(3),
  window_end timestamp(3),
```

 版权所有 © 华为技术有限公司

```
  total_num BIGINT,
  primary key (user_id, window_start) not enforced
) WITH (
  'connector' = 'jdbc',
  'url' = 'jdbc:mysql://<yourMySQL>:3306/flink',
  'table-name' = 'order_count',
  'username' = '<yourUserName>',
  'password' = '<yourPassword>',
  'sink.buffer-flush.max-rows' = '1'
);

insert into jdbcSink select
    order_id,
    TUMBLE_START(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5' SECOND),
    TUMBLE_END(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5' SECOND),
    COUNT(*) from orders
    GROUP BY user_id, TUMBLE(order_time, INTERVAL '30' SECOND, INTERVAL '10' SECOND, INTERVAL '5'
SECOND);
```

步骤4 向kafka中插入数据（这里假设同一个用户在不同时间下的订单，且因为某种原因导致10:00:13的订单数据较晚到达）：

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000002", "order_channel":"webShop", "order_time":"2021-03-24 10:00:20",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000003", "order_channel":"webShop", "order_time":"2021-03-24 10:00:33",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}

{"order_id":"202103241000000004", "order_channel":"webShop", "order_time":"2021-03-24 10:00:13",
"pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
```

步骤5 在MySQL中使用下述语句查看输出结果,，输出结果如下（因无法展示周期性输出结果，所以这里展示的是最终结果）：

```
select * from order_count
user_id     window_start         window_end         total_num
0001      2021-03-24 10:00:00  2021-03-24 10:00:30    3
0001      2021-03-24 10:00:30  2021-03-24 10:01:00    1
```

----结束

OVER WINDOW

Over Window与Group Window区别在于Over window每一行都会输出一条记录。

语法格式

```
SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  ROWS
 BETWEEN (UNBOUNDED|rowCOUNT) PRECEDING AND CURRENT ROW FROM TABLENAME

SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  RANGE
  BETWEEN (UNBOUNDED|timeInterval) PRECEDING AND CURRENT ROW FROM TABLENAME
```

语法说明

表 3-45 参数说明

参数	参数说明
PARTITION BY	指定分组的主键，每个分组各自进行计算。
ORDER BY	指定数据按processing time或event time作为时间戳。
ROWS	个数窗口。
RANGE	时间窗口。

注意事项

- 所有的聚合必须定义到同一个窗口中，即相同的分区、排序和区间。
- 当前仅支持 PRECEDING (无界或有界) 到 CURRENT ROW 范围内的窗口、FOLLOWING 所描述的区间并未支持。
- ORDER BY 必须指定于单个的时间属性。

示例

```
// 计算从规则启动到目前为止的计数及总和(in proctime)
insert into temp SELECT name,
   count(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as
cnt1,
   sum(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt2
   FROM Orders;

// 计算最近四条记录的计数及总和(in proctime)
insert into temp SELECT name,
   count(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt1,
   sum(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt2
   FROM Orders;

// 计算最近60s的计数及总和(in eventtime),基于事件时间处理，事件时间为Orders中的timeattr字段。
insert into temp SELECT name,
   count(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60'
SECOND PRECEDING AND CURRENT ROW) as cnt1,
   sum(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND
PRECEDING AND CURRENT ROW) as cnt2
   FROM Orders;
```

3.4.4 JOIN

Equi-join

语法格式

```
FROM tableExpression INNER | LEFT | RIGHT | FULL JOIN tableExpression
  ON value11 = value21 [ AND value12 = value22]
```

注意事项

- 目前仅支持 equi-join ，即 join 的联合条件至少拥有一个相等谓词。不支持任何 cross join 和 theta join。

 版权所有 © 华为技术有限公司

- Join 的顺序没有进行优化，join 会按照 FROM 中所定义的顺序依次执行。请确保 join 所指定的表在顺序执行中不会产生不支持的 cross join （笛卡儿积）以致查询失败。
- 流查询中可能会因为不同行的输入数量导致计算结果的状态无限增长。请提供具有有效保留间隔的查询配置，以防止出现过多的状态。

示例

```
SELECT *
FROM Orders INNER JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders LEFT JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders RIGHT JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders FULL OUTER JOIN Product ON Orders.productId = Product.id;
```

Time-windowed Join

功能描述

每条流的每一条数据会与另一条流上的不同时间区域的数据进行JOIN。

语法格式

```
from t1 JOIN t2 ON t1.key = t2.key AND TIMEBOUND_EXPRESSIO
```

语法描述

TIMEBOUND_EXPRESSION 有两种写法，如下：

- L.time between LowerBound(R.time) and UpperBound(R.time)
- R.time between LowerBound(L.time) and UpperBound(L.time)
- 带有时间属性(L.time/R.time)的比较表达式。

注意事项

时间窗口join需要至少一个 equi-join 谓词和一个限制了双方时间的 join 条件。

例如使用两个适当的范围谓词（<, <=, >=, >），一个 BETWEEN 谓词或一个比较两个输入表中相同类型的时间属性（即处理时间和事件时间）的相等谓词

比如，以下谓词是合法的窗口 join 条件：

- ltime = rtime
- ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
- ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND

示例

所有在收到后四小时内发货的 order 会与它们相关的 shipment 进行 join。

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.orderId AND
    o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime;
```

 版权所有 © 华为技术有限公司

Expanding arrays into a relation

注意事项

目前尚未支持非嵌套的 WITH ORDINALITY 。

示例

```
SELECT users, tag
FROM Orders CROSS JOIN UNNEST(tags) AS t (tag);
```

Join 表函数(UDTF)

功能描述

将表与表函数的结果进行 join 操作。左表（ outer ）中的每一行将会与调用表函数所产生的所有结果中相关联行进行 join 。

注意事项

针对横向表的左外部连接当前仅支持文本常量 TRUE 作为谓词。

示例

若表函数返回了空结果，左表（ outer ）的行将会被删除

```
SELECT users, tag
FROM Orders, LATERAL TABLE(unnest_udtf(tags)) t AS tag;
```

若表函数返回了空结果，将会保留相对应的外部行并用空值填充

```
SELECT users, tag
FROM Orders LEFT JOIN LATERAL TABLE(unnest_udtf(tags)) t AS tag ON TRUE;
```

Join Temporal Table Function

功能描述

注意事项

目前仅支持在 Temporal Tables 上的 inner join

示例

假如Rates是一个 Temporal Table Function， join 可以使用 SQL 进行如下的表达:

```
SELECT
  o_amount, r_rate
FROM
  Orders,
  LATERAL TABLE (Rates(o_proctime))
WHERE
  r_currency = o_currency;
```

Join Temporal Tables

功能描述

与Temporal表进行join操作

语法格式

```
SELECT column-names
FROM table1   [AS <alias1>]
```

 版权所有 © 华为技术有限公司

```
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF table1.proctime [AS <alias2>]
ON table1.column-name1 = table2.key-name1
```

语法说明

- table1.proctime表示table1的proctime处理时间属性(计算列)
- 使用FOR SYSTEM_TIME AS OF table1.proctime表示当左边表的记录与右边的维表join时，只匹配当前处理时间维表所对应的的快照数据。

注意事项

仅支持带有处理时间的 temporal tables 的 inner 和 left join

示例

假设 LatestRates 是一个根据最新的 rates 物化的Temporal Table。

```
SELECT
  o.amout, o.currency, r.rate, o.amount * r.rate
FROM
  Orders AS o
  JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
  ON r.currency = o.currency;
```

3.4.5 OrderBy & Limit

OrderBy

功能描述

主要根据时间属性按照升序进行排序

注意事项

目前仅支持根据时间属性进行排序

示例

对订单根据订单时间进行升序排序

```
SELECT *
FROM Orders
ORDER BY orderTime;
```

Limit

功能描述

限制返回的数据结果个数

注意事项

LIMIT查询需要有一个ORDER BY

示例

```
SELECT *
FROM Orders
ORDER BY orderTime
LIMIT 3;
```

 版权所有 © 华为技术有限公司

3.4.6 Top-N

功能描述

Top-N 查询是根据列排序找到N个最大或最小的值。最大值集和最小值集都被视为是一种 Top-N 的查询。若在批处理或流处理的表中需要显示出满足条件的 N 个最底层记录或最顶层记录， Top-N 查询将会十分有用。

语法格式

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     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]
```

语法说明

- ROW_NUMBER(): 根据当前分区内的各行的顺序从第一行开始，依次为每一行分配一个唯一且连续的号码。目前，我们只支持 ROW_NUMBER 在 over 窗口函数中使用。未来将会支持 RANK() 和 DENSE_RANK()函数。
- PARTITION BY col1[, col2...]: 指定分区列，每个分区都将会有一个 Top-N 结果。
- ORDER BY col1 [asc|desc][, col2 [asc|desc]...]: 指定排序列，不同列的排序方向可以不一样。
- WHERE rownum <= N: Flink 需要 rownum <= N 才能识别一个查询是否为 Top-N 查询。 其中， N 代表最大或最小的 N 条记录会被保留。
- [AND conditions]: 在 where 语句中，可以随意添加其他的查询条件，但其他条件只允许通过 AND 与 rownum <= N 结合使用。

注意事项

- TopN 查询的结果会带有更新。
- Flink SQL 会根据排序键对输入的流进行排序。
- 如果 top N 的记录发生了变化，变化的部分会以撤销、更新记录的形式发送到下游。
- 如果 top N 记录需要存储到外部存储，则结果表需要拥有相同与 Top-N 查询相同的唯一键。

示例

查询每个分类实时销量最大的五个产品

```
SELECT *
  FROM (
     SELECT *,
         ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) as row_num
     FROM ShopSales)
  WHERE row_num <= 5;
```

 版权所有 © 华为技术有限公司

3.4.7 去重

功能描述

对在列的集合内重复的行进行删除，只保留第一行或最后一行数据。

语法格式

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]
       ORDER BY time_attr [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1
```

语法说明

- ROW_NUMBER(): 从第一行开始，依次为每一行分配一个唯一且连续的号码。
- PARTITION BY col1[, col2...]: 指定分区的列，例如去重的键。
- ORDER BY time_attr [asc|desc]: 指定排序的列。所指定的列必须为时间属性。目前仅支持proctime。升序（ ASC ）排列指只保留第一行，而降序排列（ DESC ）则只保留最后一行。
- WHERE rownum = 1: Flink 需要 rownum = 1 以确定该查询是否为去重查询。

注意事项

无

示例

根据order_id对数据进行去重，其中proctime为事件时间属性列

```
SELECT order_id, user, product, number
  FROM (
    SELECT *,
       ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY proctime ASC) as row_num
    FROM Orders)
  WHERE row_num = 1;
```

3.5 函数

3.5.1 自定义函数

概述

DLI支持三种自定义函数：

- UDF：自定义函数，支持一个或多个输入参数，返回一个结果值。
- UDTF：自定义表值函数，支持一个或多个输入参数，可返回多行多列。
- UDAF：自定义聚合函数，将多条记录聚合成一个值。

 版权所有 © 华为技术有限公司

说明

- 暂不支持通过python写UDF、UDTF、UDAF自定义函数。
- Flink Opensource SQL作业中使用自定义函数时，不支持生成静态流图。

POM 依赖

```
<dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-table-common</artifactId>
      <version>1.10.0</version>
      <scope>provided</scope>
</dependency>
```

使用方式

1. 将写好的自定义函数打成JAR包，并上传到OBS上。
2. 在DLI管理控制台的左侧导航栏中，单击数据管理>“程序包管理”，然后单击创建，并使用OBS中的jar包创建相应的程序包。
3. 在DLI管理控制台的左侧导航栏中，单击作业管理>“Flink作业”，在需要编辑作业对应的“操作”列中，单击“编辑”，进入作业编辑页面。
4. 在“运行参数设置”页签，“UDF Jar”选择创建的程序包，单击“保存”。
5. 选定JAR包以后，SQL里添加UDF声明语句，就可以像普通函数一样使用了。

   ```
   CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
   ```

UDF

UDF函数需继承ScalarFunction函数，并实现eval方法。open函数及close函数可选。

编写代码示例

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class UdfScalarFunction extends ScalarFunction {
  private int factor = 12;
  public UdfScalarFunction() {
    this.factor = 12;
  }
  /**
   * 初始化操作，可选
   * @param context
   */
  @Override
  public void open(FunctionContext context) {}
  /**
   * 自定义逻辑
   * @param s
   * @return
   */
   public int eval(String s) {
     return s.hashCode() * factor;
   }
   /**
    * 可选
    */
   @Override
   public void close() {}
}
```

使用示例

```
CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
INSERT INTO sink_stream select udf_test(attr) FROM source_stream;
```

UDTF

UDTF函数需继承TableFunction函数，并实现eval方法。open函数及close函数可选。如果需要UDTF返回多列，只需要将返回值声明成Tuple或Row即可。若使用Row，需要重载getResultType声明返回的字段类型。

编写代码示例

```
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class UdfTableFunction extends TableFunction<Row> {
  private Logger log = LoggerFactory.getLogger(TableFunction.class);
  /**
   * 初始化操作，可选
   * @param context
   */
  @Override
  public void open(FunctionContext context) {}
  public void eval(String str, String split) {
    for (String s : str.split(split)) {
      Row row = new Row(2);
      row.setField(0, s);
      row.setField(1, s.length());
      collect(row);
    }
  }
  /**
   * 函数返回类型声明
   * @return
   */
  @Override
  public TypeInformation<Row> getResultType() {
  return Types.ROW(Types.STRING, Types.INT);
  }
  /**
   * 可选
   */
  @Override
  public void close() {}
}
```

使用示例

UDTF支持CROSS JOIN和LEFT JOIN，在使用UDTF时需要带上 LATERAL 和TABLE 两个关键字。

- CROSS JOIN：对于左表的每一行数据，假设UDTF不产生输出，则这一行不进行输出。
- LEFT JOIN：对于左表的每一行数据，假设UDTF不产生输出，这一行仍会输出，UDTF相关字段用null填充。

```
CREATE FUNCTION udtf_test AS 'com.huaweicompany.udf.TableFunction';
// CROSS JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length);
// LEFT JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

 版权所有 © 华为技术有限公司

UDAF

UDAF函数需继承AggregateFunction函数。首先需要创建一个用来存储计算结果的Accumulator，如示例里的WeightedAvgAccum。

编写代码示例

```
public class WeightedAvgAccum {
public long sum = 0;
public int count = 0;
}
```

```
import org.apache.flink.table.functions.AggregateFunction;
import java.util.Iterator;
/**
* 第一个类型变量为聚合函数返回的类型，第二个类型变量为Accumulator类型
* Weighted Average user-defined aggregate function.
*/
public class UdfAggFunction extends AggregateFunction<Long, WeightedAvgAccum> {
  // 初始化Accumulator
  @Override
  public WeightedAvgAccum createAccumulator() {
    return new WeightedAvgAccum();
  }
  // 返回Accumulator存储的中间计算值
  @Override
  public Long getValue(WeightedAvgAccum acc) {
    if (acc.count == 0) {
       return null;
    } else {
      return acc.sum / acc.count;
 }
}
// 根据输入更新中间计算值
public void accumulate(WeightedAvgAccum acc, long iValue) {
acc.sum += iValue;
acc.count += 1;
}
// Restract撤回操作，和accumulate操作相反
public void retract(WeightedAvgAccum acc, long iValue) {
acc.sum -= iValue;
acc.count -= 1;
}
// 合并多个accumulator值
public void merge(WeightedAvgAccum acc, Iterable<WeightedAvgAccum> it) {
Iterator<WeightedAvgAccum> iter = it.iterator();
while (iter.hasNext()) {
WeightedAvgAccum a = iter.next();
acc.count += a.count;
acc.sum += a.sum;
}
}
// 重置中间计算值
public void resetAccumulator(WeightedAvgAccum acc) {
acc.count = 0;
acc.sum = 0L;
}
}
```

使用示例

```
CREATE FUNCTION udaf_test AS 'com.huaweicompany.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

 版权所有 © 华为技术有限公司

3.5.2 自定义函数类型推导

操作场景

类型推导包含了验证输入值、派生参数和返回值数据类型。从逻辑角度看，Planner需要知道数据类型、精度和小数位数；从 JVM 角度来看，Planner 在调用自定义函数时需要知道如何将内部数据结构表示为JVM对象。

Flink 自定义函数实现了自动的类型推导提取，通过反射从函数的类及其求值方法中派生数据类型。然而以反射方式提取数据类型并不总是成功的，比如UDTF中常见的Row类型。

由于 Flink 1.11 起引入了新的自定义函数注册接口，使用了新的自定义函数类型推断机制，因此原先1.10 重载 getResultType 声明返回字段类型的方式将不再可用。继续使用会抛出如下异常：

```
Caused by: org.apache.flink.table.api.ValidationException: Cannot extract a data type from a pure 'org.apache.flink.types.Row' class. Please use annotations to define field names and field types.
```

目前 Flink 1.12 可以通过使用DataTypeHint 和FunctionHint 注解相关参数、类或方法来支持提取过程。

代码示例

Table（类似于 SQL 标准）是一种强类型的 API，函数的参数和返回类型都必须映射到 Table API 的数据类型，参见**Table API数据类型**。

如果需要更高级的类型推导逻辑，您可以在每个自定义函数中显式重写 getTypeInference() 方法。

建议使用注解方式，因为它可使自定义类型推导逻辑保持在受影响位置附近，而在其他位置则保持默认状态。

```
importorg.apache.flink.table.annotation.DataTypeHint;
importorg.apache.flink.table.annotation.FunctionHint;
importorg.apache.flink.table.functions.FunctionContext;
importorg.apache.flink.table.functions.TableFunction;
importorg.apache.flink.types.Row;
publicclassUdfTableFunctionextendsTableFunction<Row>{
  /**
   *初始化操作，可选
   *@paramcontext
   */
  @Override
   public void open(FunctionContextcontext) {  }

  @FunctionHint(output=@DataTypeHint("ROW<s STRING, i INT>"))
  publicvoideval(String str, String split) {
      for (String s: str.split(split)) {
          Row row=new Row(2);
          row.setField(0, s);
          row.setField(1, s.length());
          collect(row);
      }
  }
  /**
  * 可选
  */
 @Override
 public void close() {}
}
```


版权所有 © 华为技术有限公司

使用示例

UDTF支持CROSS JOIN和LEFT JOIN，在使用UDTF时需要带上 LATERAL 和TABLE 两个关键字。

- CROSS JOIN：对于左表的每一行数据，假设UDTF不产生输出，则这一行不进行输出。
- LEFT JOIN：对于左表的每一行数据，假设UDTF不产生输出，这一行仍会输出，UDTF相关字段用null填充。

```
CREATE FUNCTION udtf_test AS 'com.huaweicompany.udf.TableFunction';-- CROSS JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length);-- LEFT JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN
LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

3.5.3 自定义函数参数传递

操作场景

如果您的自定义函数需要在多个作业中使用，但对于不同作业某些参数值不同，直接在UDF中修改较为复杂。您可以在Flink OpenSource SQL编辑页面，自定义配置中配置参数**pipeline.global-job-parameters**，在UDF代码中获取该参数并使用。如需修改参数值，直接在FlinkOpenSource SQL编辑页面，自定义配置中修改该参数值，即可达到快速修改UDF参数值的目的。

操作步骤

自定义函数中提供了可选的open(FunctionContext context)方法，FunctionContext具备参数传递功能，自定义配置项通过此对象来传递。自定义函数的参数传递操作步骤如下：

1. 在Flink OpenSource SQL编辑页面右侧自定义配置中添加参数pipeline.global-job-parameters，格式如下：
   ```
   pipeline.global-job-parameters=k1:v1,"k2:v1,v2",k3:"str:ing","k4:str""ing"
   ```
   该配置定义了如表3-46的map。

   表 3-46 pipeline.global-job-parameters 示例

key	value
k1	v1
k2	v1,v2
k3	str:ing
k4	str""ing

版权所有 © 华为技术有限公司

说明

- FunctionContext#getJobParameter只能获取pipeline.global-job-parameters这一配置项的值。因此需要将UDF用到的所有配置项全部写入到pipeline.global-job-parameters中。
- key和value之间通过冒号(:)分隔，所有key-value用逗号(,)连接。
- 如果key或value中含有逗号(,)，则需要用双引号(")将key:value整个包围起来。参考k2。
- 如果key或value中含有半角冒号(:)，则需要用双引号(")将key或value包围起来。参考k3。
- 如果key或value中含有双引号(")，则需要通过连写两个双引号("")进行转义，也需要用双引号(")将key:value整个包围起来。参考k4。

2. 在自定义函数代码中，通过FunctionContext#getJobParameter获取map的各项内容，代码示例如下：

```
context.getJobParameter("url","jdbc:mysql://xx.xx.xx.xx:3306/table");
context.getJobParameter("driver","com.mysql.jdbc.Driver");
context.getJobParameter("user","user");
context.getJobParameter("password","password");
```

代码示例

以下是一个UDF示例：通过pipeline.global-job-parameters传入连接数据库需要的url、user、password等参数，获取udf_info表数据后和流数据拼接成json输出。

表 3-47 udf_info 表

key	value
class	class-4

SimpleJsonBuild.java

```
package udf;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.HashMap;
import java.util.Map;

public class SimpleJsonBuild extends ScalarFunction {
  private static final Logger LOG = LoggerFactory.getLogger(SimpleJsonBuild.class);
  String remainedKey;
  String remainedValue;

  private Connection initConnection(Map<String, String> userParasMap) {
     String url = userParasMap.get("url");
     String driver = userParasMap.get("driver");
     String user = userParasMap.get("user");
     String password = userParasMap.get("password");
```

```
    Connection conn = null;
    try {
        Class.forName(driver);
        conn = DriverManager.getConnection(url, user, password);
        LOG.info("connect successfully");
    } catch (Exception e) {
        LOG.error(String.valueOf(e));
    }
    return conn;
}

@Override
public void open(FunctionContext context) throws Exception {
    Map<String, String> userParasMap = new HashMap<>();
    Connection connection;
    PreparedStatement pstmt;
    ResultSet rs;

    String url = context.getJobParameter("url","jdbc:mysql://xx.xx.xx.xx:3306/table");
    String driver = context.getJobParameter("driver","com.mysql.jdbc.Driver");
    String user = context.getJobParameter("user","user");
    String password = context.getJobParameter("password","password");

    userParasMap.put("url", url);
    userParasMap.put("driver", driver);
    userParasMap.put("user", user);
    userParasMap.put("password", password);

    connection = initConnection(userParasMap);
    String sql = "select `key`, `value` from udf_info";
    pstmt = connection.prepareStatement(sql);
    rs = pstmt.executeQuery();

    while (rs.next()) {
        remainedKey = rs.getString(1);
        remainedValue = rs.getString(2);
    }
}

public String eval(String... params) throws IOException {
    if (params != null && params.length != 0 && params.length % 2 <= 0) {
        HashMap<String, String> hashMap = new HashMap();
        for (int i = 0; i < params.length; i += 2) {
            hashMap.put(params[i], params[i + 1]);
            LOG.debug("now the key is " + params[i].toString() + "; now the value is " + params[i +
1].toString());
        }
        hashMap.put(remainedKey, remainedValue);
        ObjectMapper mapper = new ObjectMapper();
        String result = "{}";
        try {
            result = mapper.writeValueAsString(hashMap);
        } catch (Exception ex) {
            LOG.error("Get result failed." + ex.getMessage());
        }
        LOG.debug(result);
        return result;
    } else {
        return "{}";
    }
}

public static void main(String[] args) throws IOException {
    SimpleJsonBuild  sjb = new SimpleJsonBuild();
    System.out.println(sjb.eval("json1", "json2", "json3", "json4"));
}
}
```

 版权所有 © 华为技术有限公司

在Flink OpenSource SQL编辑页面右侧**自定义配置**中添加参数pipeline.global-job-parameters

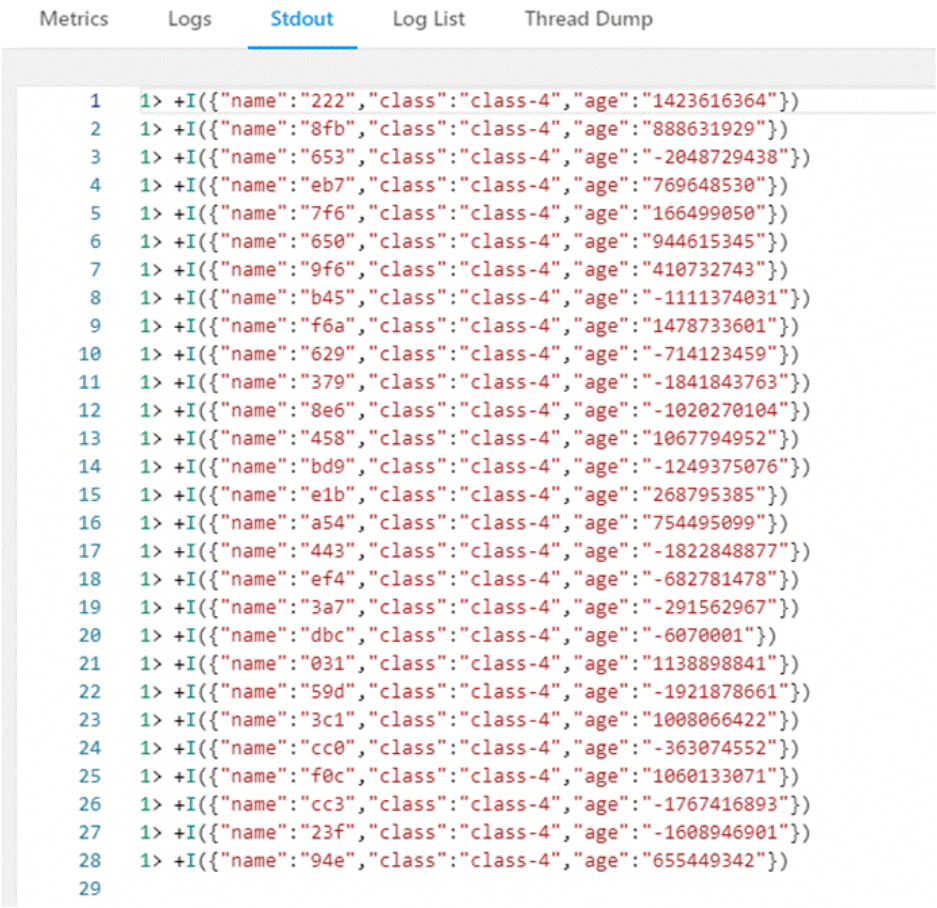
```
pipeline.global-job-parameters=url:'jdbc:mysql://x.x.x.x:xxxx/
swqtest',driver:com.mysql.jdbc.Driver,user:xxx,password:xxx
```

Flink OpenSource SQL

```
create function SimpleJsonBuild AS 'udf.SimpleJsonBuild';
create table dataGenSource(user_id string, amount int) with (
 'connector' = 'datagen',
 'rows-per-second' = '1', --每秒生成一条数据
 'fields.user_id.kind' = 'random', --为字段user_id指定random生成器
 'fields.user_id.length' = '3' --限制user_id长度为3
);
create table printSink(message STRING) with ('connector' = 'print');
insert into
printSink
SELECT
SimpleJsonBuild("name", user_id, "age", cast(amount as string))
from
dataGenSource;
```

运行结果

单击Flink作业操作列下的“更多 > FlinkUI > Task Managers > Stdout”查看输出结果:



3.5.4 内置函数

版权所有 © 华为技术有限公司

3.5.4.1 数学运算函数

关系运算符

所有数据类型都可用关系运算符进行比较，并返回一个BOOLEAN类型的值。

关系运算符均为双目操作符，被比较的两个数据类型必须是相同的数据类型或者是可以进行隐式转换的类型。

Flink SQL提供的关系运算符，请参见表3-48。

表 3-48 关系运算符

运算符	返回类型	描述
A = B	BOOLEAN	若A与B相等，返回TRUE，否则返回FALSE。用于做赋值操作。
A <> B	BOOLEAN	若A与B不相等，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL，该种运算符为标准SQL语法。
A < B	BOOLEAN	若A小于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A <= B	BOOLEAN	若A小于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A > B	BOOLEAN	若A大于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A >= B	BOOLEAN	若A大于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A IS NULL	BOOLEAN	若A为NULL则返回TRUE，否则返回FALSE。
A IS NOT NULL	BOOLEAN	若A不为NULL，则返回TRUE，否则返回FALSE。
A IS DISTINCT FROM B	BOOLEAN	若A与B不相等，则返回TRUE，将空值视为相同。
A IS NOT DISTINCT FROM B	BOOLEAN	若A与B相等，则返回TRUE，将空值视为相同。
A BETWEEN [ASYMMETRIC | SYMMETRIC] B AND C	BOOLEAN	若A大于或等于B且小于或等于C，则返回TRUE。 • ASYMMETRIC：表示B和C位置相关。 例如：A BETWEEN ASYMMETRIC B AND C 等价于 (A BETWEEN B AND C)。 • SYMMETRIC：表示B和C位置不相关。 例如：A BETWEEN SYMMETRIC B AND C 等价于 (A BETWEEN B AND C) OR (A BETWEEN C AND B)。

版权所有 © 华为技术有限公司

运算符	返回类型	描述
A NOT BETWEEN B [ASYMMETRIC | SYMMETRIC]AND C	BOOLEAN	若A小于B或大于C，则返回TRUE。 ● ASYMMETRIC：表示B和C位置相关。 例如：A NOT BETWEEN ASYMMETRIC B AND C 等价于 (A NOT BETWEEN B AND C)。 ● SYMMETRIC：表示B和C位置不相关。 例如：A NOT BETWEEN SYMMETRIC B AND C 等价于 (A NOT BETWEEN B AND C) OR (A NOT BETWEEN C AND B)。
A LIKE B [ESCAPE C]	BOOLEAN	若A与模式B匹配，则返回TRUE。 必要时可以定义转义字符C。
A NOT LIKE B [ESCAPE C]	BOOLEAN	若A与模式B不匹配，则返回TRUE。 必要时可以定义转义字符C。
A SIMILAR TO B [ESCAPE C]	BOOLEAN	若A与正则表达式B匹配，则返回TRUE。 必要时可以定义转义字符C。
A NOT SIMILAR TO B [ESCAPE C]	BOOLEAN	若A与正则表达式B不匹配，则返回TRUE。 必要时可以定义转义字符C。
value IN (value [, value]*)	BOOLEAN	若值等于列表中的值，则返回TRUE。
value NOT IN (value [, value]*)	BOOLEAN	若值不等于列表中的每个值，则返回TRUE。
EXISTS (sub-query)	BOOLEAN	若子查询至少返回一条数据，则返回TRUE。
value IN (sub-query)	BOOLEAN	若值等于子查询返回的某个值，则返回TRUE。
value NOT IN (sub-query)	BOOLEAN	若值不等于子查询返回的每个值，则返回TRUE。

注意事项

- double、real和float值存在一定的精度差。且我们不建议直接使用等号“=”对两个double类型数据进行比较。用户可以使用两个double类型相减，而后取绝对值的方式判断。当绝对值足够小时，认为两个double数值相等，例如:
  ```
  abs(0.9999999999 - 1.0000000000) < 0.000000001 //0.9999999999和1.0000000000为10位精度，而0.000000001为9位精度，此时可以认为0.9999999999和1.0000000000相等。
  ```
- 数值类型可与字符串类型进行比较。做大小(>,<,>=,<=)比较时，会默认将字符串转换为数值类型，因此不支持字符串内有除数字字符之外的字符。
- 字符串之间可以进行比较。

 版权所有 © 华为技术有限公司

逻辑运算符

常用的逻辑操作符有AND、OR和NOT，优先级顺序为：NOT>AND>OR。

运算规则请参见**表3-49**，表中的A和B代表逻辑表达式。

表 3-49 逻辑运算符

运算符	返回类型	描述
A OR B	BOOLEAN	若A或B为TRUE，则返回TRUE，且支持三值逻辑。
A AND B	BOOLEAN	若A和B为TRUE，则返回TRUE，且支持三值逻辑。
NOT A	BOOLEAN	若A不为TRUE则返回TRUE；若A为UNKNOWN，返回UNKNOWN。
A IS FALSE	BOOLEAN	若A为FALSE则返回TRUE； 若A为UNKNOWN，则返回FALSE。
A IS NOT FALSE	BOOLEAN	若A不为FALSE则返回TRUE；若A为UNKNOWN，则返回TRUE。
A IS TRUE	BOOLEAN	若A为TRUE，则返回TRUE；若A为UNKNOWN，则返回FALSE。
A IS NOT TRUE	BOOLEAN	若A不为TRUE则返回TRUE；若A为UNKNOWN，则返回TRUE。
A IS UNKNOWN	BOOLEAN	若A为UNKNOWN，则返回TRUE。
A IS NOT UNKNOWN	BOOLEAN	若A不为UNKNOWN，则返回TRUE。

注意事项

逻辑操作符只允许boolean类型参与运算，不支持隐式类型转换。

算术运算符

算术运算符包括双目运算符与单目运算符，这些运算符都将返回数字类型。Flink SQL所支持的算术运算符如**表3-50**所示。

表 3-50 算术运算符

运算符	返回类型	描述
+ numeric	所有数字类型	返回数字。

 版权所有 © 华为技术有限公司

运算符	返回类型	描述
- numeric	所有数字类型	返回负数。
A + B	所有数字类型	A和B相加。结果数据类型与操作数据类型相关，例如一个整数类型数据加上一个浮点类型数据，结果数值为浮点类型数据。
A - B	所有数字类型	A和B相减。结果数据类型与操作数据类型相关。
A * B	所有数字类型	A和B相乘。结果数据类型与操作数据类型相关。
A / B	所有数字类型	A和B相除。结果是一个double（双精度）类型的数值。
POWER(A, B)	所有数字类型	返回A数的B次方乘幂。
ABS(numeric)	所有数字类型	返回数值的绝对值。
MOD(A, B)	所有数字类型	返回A除以B的余数（模数）。 返回值只有在A为负数时才为负数。
SQRT(A)	所有数字类型	返回A的平方根。
LN(A)	所有数字类型	返回A的自然对数（基数e）。
LOG10(A)	所有数字类型	返回A的基数10对数。
LOG2(A)	所有数字类型	返回A的基数2对数。
LOG(B) LOG(A, B)	所有数字类型	当只有一个参数，返回B的自然对数（基数e）。 当有两个参数，返回B以A为基数的对数。 B必须大于0，且A必须大于1。

版权所有 © 华为技术有限公司

运算符	返回类型	描述
EXP(A)	所有数字类型	返回e的a次方。
CEIL(A) CEILING(A)	所有数字类型	将参数向上舍入为最接近的整数。例如ceil(21.2)，返回22。
FLOOR(A)	所有数字类型	对给定数据进行向下舍入最接近的整数。例如floor(21.2)，返回21。
SIN(A)	所有数字类型	计算给定A的正弦值。
COS(A)	所有数字类型	计算给定A的余弦值。
TAN(A)	所有数字类型	计算给定A的正切值。
COT(A)	所有数字类型	计算给定A的余切值。
ASIN(A)	所有数字类型	计算给定A的反正弦值。
ACOS(A)	所有数字类型	计算给定A的反余弦值。
ATAN(A)	所有数字类型	计算给定A的反正切值。
ATAN2(A, B)	所有数字类型	计算给定坐标(A, B)的反正切值。
COSH(A)	所有数字类型	计算给定A的双曲余弦值。返回类型为DOUBLE。
DEGREES(A)	所有数字类型	返回弧度所对应的角度。

版权所有 © 华为技术有限公司

运算符	返回类型	描述
RADIANS(A)	所有数字类型	返回角度所对应的弧度。
SIGN(A)	所有数字类型	返回a所对应的正负号，a为正返回1，a为负，返回-1，否则返回0。
ROUND(A, d)	所有数字类型	返回小数部分，d位之后数字的四舍五入，d为int型。例如round(21.263,2)，返回21.26。
PI	所有数字类型	返回pi的值。
E()	所有数字类型	返回e的值。
RAND()	所有数字类型	返回一个0.0和1.0之间的随机double类型的数（包含0.0，不包含1.0）。
RAND(A)	所有数字类型	根据初始化种子A，返回一个0.0和1.0之间的随机double类型的数（包含0.0，不包含1.0）。若初始化种子相同，则返回的随机数相同。
RAND_INTEGER(A)	所有数字类型	返回一个0和A之间的随机整数（包含0，不包含A）。
RAND_INTEGER(A, B)	所有数字类型	根据初始化种子A，返回一个0和B之间的随机整数值（包含0，不包含B）
UUID()	所有数字类型	返回一个UUID字符串。
BIN(A)	所有数字类型	返回一个整数A的二进制字符串。如为null则返回null。
HEX(A) HEX(B)	所有数字类型	返回一个整数A或者字符串B的十六进制字符串。若A或B为null，则返回null。
TRUNCATE(A, d)	所有数字类型	返回保留小数点后d为小数的数字。若A或d为null，则返回null。 例如：truncate(42.345, 2) = 42.340 truncate(42.345) = 42.000

版权所有 © 华为技术有限公司

运算符	返回类型	描述
PI()	所有数字类型	返回pi的值

注意事项

字符串类型不能参与算术运算。

3.5.4.2 字符串函数

字符串函数

数据湖探索（DLI）提供了丰富的字符串函数，用于处理和转换字符串数据。这些函数包括拼接、大小写转换、截取子串、替换、正则匹配、编码解码、格式转换等。此外，还支持字符串长度计算、位置查找、填充、反转等功能，以及从JSON字符串中提取值的JSON_VAL函数。这些功能广泛应用于数据清洗、文本处理和数据分析场景，为开发者提供强大的工具支持。

字符串函数简介请参考**表3-51**，更多内容参考**Apache Flink**。

表 3-51 字符串函数

函数	返回类型	描述
string1 || string2	STRING	返回两个字符串的拼接
CHAR_LENGTH(string) CHARACTER_LENGTH(string)	INT	返回字符串中的字符数量
UPPER(string)	STRING	返回字符串的大写形式
LOWER(string)	STRING	返回字符串的小写形式
POSITION(string1 IN string2)	INT	返回第一个字符串在第二个字符串中首次出现的位置。若第一个字符串不存在与第二个字符串，则返回0
TRIM([BOTH | LEADING | TRAILING] string1 FROM string2)	STRING	去除string2字符串的首尾(或首部、或尾部)的string1字符串
LTRIM(string)	STRING	返回去除首部空格后的字符串 例如LTRIM(' This is a test String.') 返回"This is a test String."

版权所有 © 华为技术有限公司

函数	返回类型	描述
RTRIM(string)	STRING	返回去除尾部空格后的字符串 例如RTRIM('This is a test String. ') 返回"This is a test String."
REPEAT(string, integer)	STRING	返回integer个string连接后的字符串 例如REPEAT('This is a test String.', 2) 返回"This is a test String.This is a test String."
REGEXP_REPLACE(string1, string2, string3)	STRING	用string3代替string1中的符合正则表达式string2的字符串，并返回替换后的string1字符串 例如REGEXP_REPLACE('foobar', 'oo|ar', '') 返回 "fb" REGEXP_REPLACE('ab\ab', '\\', 'e')返回"abeab"
OVERLAY(string1 PLACING string2 FROM integer1 [FOR integer2])	STRING	用string2代替string1中的字符串，从integer1开始，替换长度为integer2，并返回替换后的string1字符串 integer2默认为string2的长度 例如OVERLAY('This is an old string' PLACING ' new' FROM 10 FOR 5)返回"This is a new string"
SUBSTRING(string FROM integer1 [FOR integer2])	STRING	返回string中从integer1位置开始的长度为integer2的子字符串。若integer2未配置，则默认返回从integer1开始到末尾的子字符串
REPLACE(string1, string2, string3)	STRING	用string3代替string1中的string2后的字符串，并返回替换后的string1字符串 例如：REPLACE('hello world', 'world', 'flink') 返回 "hello flink" REPLACE('abababb', 'abab', 'z') 返回 "zab" REPLACE('ab\\ab', '\\', 'e')返回"abeab"
REGEXP_EXTRACT(string1, string2[, integer])	STRING	使用正则表达式string2匹配抽取字符串string1中的第integer个字串，integer从1开始，正则匹配提取。 若参数为 NULL或者正则不合法，则返回NULL。 例如REGEXP_EXTRACT('foothebar', 'foo(.*?)(bar)', 2)" 返回"bar"
INITCAP(string)	STRING	返回将字符串的首字符大写其余字符转为小写后的字符串
CONCAT(string1, string2,...)	STRING	返回将两个或多个字符串拼接后的新字符串。 例如 CONCAT('AA', 'BB', 'CC') 返回"AABBCC"

版权所有 © 华为技术有限公司

函数	返回类型	描述
CONCAT_WS(string1, string2, string3,...)	STRING	返回将每个参数和第一个参数指定的分隔符依次连接到一起组成的字符串。若string1是null，则返回null。若其他参数为null，在执行拼接过程中跳过取值为null的参数 例如CONCAT_WS('~', 'AA', NULL, 'BB', '', 'CC') 返回"AA~BB~~CC"
LPAD(string1, integer, string2)	STRING	将string2字符串拼接到string1字符串的左端，直到新的字符串达到指定长度integer为止 任意参数为null时，返回null 若integer为负数，则返回null 若integer不大于string1的长度，则返回string1裁剪为integer长度的字符串 例如LPAD('hi',4,'??') 返回"??hi" LPAD('hi',1,'??') 返回"h"
RPAD(string1, integer, string2)	STRING	将string2字符串拼接到string1字符串的右端，直到新的字符串达到指定长度integer为止 任意参数为null时，返回null 若integer为负数，则返回null 若integer不大于string1的长度，则返回string1裁剪为integer长度的字符串 例如RPAD('hi',4,'??') 返回 "hi??" RPAD('hi',1,'??') 返回"h"
FROM_BASE64(string)	STRING	将base64编码的字符串str解析成对应字符串 若字符串为null，则返回null 例如FROM_BASE64('aGVsbG8gd29ybGQ=') 返回"hello world"
TO_BASE64(string)	STRING	将字符串基于base64编码 若字符串为null，则返回null 例如TO_BASE64('hello world') 返回"aGVsbG8gd29ybGQ="
ASCII(string)	INT	返回字符串的第一个字符的ASCII值 若字符串为null，则返回null 例如ascii('abc') 返回97 ascii(CAST(NULL AS VARCHAR)) 返回NULL

版权所有 © 华为技术有限公司

函数	返回类型	描述
CHR(integer)	STRING	将ASCII码转换为字符 若integer大于255，则计算出integer除以255的余数，并将余数作为ASCII码值 若integer为null，则返回null chr(97) 返回a chr(353) 返回a
DECODE(binary, string)	STRING	使用提供的字符集string解码参数binary，字符集可以为'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16' 若任意参数为null，则返回null
ENCODE(strinh1, string2)	STRING	使用提供的字符集string2编码字符串string1，字符集可以为'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16' 若任意参数为null，则返回null
INSTR(string1, string2)	INT	返回string2在string1中首次出现的位置 若有参数为null，则返回null
LEFT(string, integer)	STRING	返回最左边的integer个字符 若integer为负数，则返回空 若存在参数为null，则返回null
RIGHT(string, integer)	STRING	返回最右侧的integer个字符 若integer为负数，则返回空 若存在参数为null，则返回null
LOCATE(string1, string2[, integer])	INT	返回string1在string2的位置integer之后首次出现的位置 若string1在string2的位置integer之后不存在，则返回0 若integer不存在，则默认为0 若存在参数为null，则返回null

版权所有 © 华为技术有限公司

函数	返回类型	描述
PARSE_URL(string1, string2[, string3])	STRING	返回URL string1中指定的部分解析后的值 string2为'HOST'、'PATH'、'QUERY'、'REF'、'PROTOCOL'、'AUTHORITY'、'FILE'或'USERINFO' 若存在参数为null，则返回null 若string2为QUERY，也可以指定QUERY中的key为string3 例如： parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST')返回 'facebook.com' parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') 返回'v1'
REGEXP(string1, string2)	BOOLEAN	对指定的字符串执行一个正则表达式搜索，并返回一个BOOLEAN值表示是否找到指定的匹配模式。若找到，则返回TRUE。其中string1表示指定的字符串，string2表示正则表达式 若存在参数为null，则返回null
REVERSE(string)	STRING	反转字符串，返回字符串值的相反顺序。 若存在参数为null，则返回null **说明** 使用时请注意需在函数上加反引号 ` REVERSE ` 。
SPLIT_INDEX(string1, string2, integer1)	STRING	以string2作为分隔符，将字符串string1分割成若干段，取其中的第integer1段。integer1从0开始 若integer1为负数，则返回null 如果任一参数为null，则返回null 具体函数说明请参考**SPLIT_INDEX 函数说明**。
STR_TO_MAP(string1[, string2, string3]])	MAP	使用string2分隔符将string1分割成K-V对，并使用string3分隔每个K-V对，组装成MAP返回 string2默认为',' string3默认为'='
SUBSTR(string[, integer1[, integer2]])	STRING	截取从位置integer1开始，长度为integer2的子串，并返回 若为指定integer2，翻截取到字符串结尾

版权所有 © 华为技术有限公司

函数	返回类型	描述
JSON_VAL(STRING json_string, STRING json_path)	STRING	从json形式的字符串json_string中提取指定json_path的值。具体函数使用可以参考**JSON_VAL函数说明**说明。 **说明** 以下规则优先级按照顺序从高到低。 1. 不允许json_string和json_path为NULL 2. json_string格式必须为合法的json串，否则函数返回NULL 3. json_string为空字符串，则函数返回空字符串 4. json_path为空字串或路径不存在，则函数返回NULL

SPLIT_INDEX 函数说明

SPLIT_INDEX(string1, string2, integer1)

- **参数说明**
  - **string1：**
    - 类型：STRING
    - 说明：需要被分割的原始字符串。
    - 示例：'a,b,c,d' 或 'a\bc\bd'
  - **string2：**
    - 类型：STRING
    - 说明：分割字符串的分隔符。
    - 特殊字符处理：
      - 如果分隔符中包含特殊字符（如 \\、* 等），需要使用双反斜杠 \\ 进行转义。
      - 分隔符为 . 时不需要转义。
      - 如果分隔符为 \\ 本身，应写为 '\\\\'。
  - **integer1：**
    - 类型：INT
    - 说明：指定提取的分割后的部分索引，从 0 开始计数。
    - 示例：0 表示提取第一个部分，1 表示提取第二个部分，依此类推。
- **返回值**

  返回分割后的字符串中指定索引位置的部分。

  如果索引超出范围或输入参数为 NULL，返回 NULL。
- **示例**1

  SELECT SPLIT_INDEX('a,b,c,d', ',', 1); -- 返回 'b'

版权所有 © 华为技术有限公司

- **示例2：**
  点号（.）作为分隔符，不需要转义：
  SELECT SPLIT_INDEX('a.b.c.d', '.', 2); -- 返回 'c'
- **示例3：**
  反斜杠（\）作为分隔符：
  反斜杠本身也需要转义，使用 '\\\\'：
  SELECT SPLIT_INDEX('a\\bc\\bd', '\\\\', 1); -- 返回 'bc'

JSON_VAL 函数说明

- 语法

```
STRING JSON_VAL(STRING json_string, STRING json_path)
```

表 3-52 参数说明

参数	数据类型	说明
json_string	STRING	需要解析的JSON对象，使用字符串表示。
json_path	STRING	解析JSON的路径表达式，使用字符串表示。 目前path支持如下表达式参考下表**表 3-53**。

表 3-53 json_path 参数支持的表达式

表达式	说明
$	根对象
[]	数组下标
*	数组通配符
.	取子元素

- 示例
  a. 测试输入数据。
  测试数据源kafka，具体消息内容参考如下：

```
{name:James,age:24,gender:male,grade:{math:95,science:[80,85],english:100}}
{name:James,age:24,gender:male,grade:{math:95,science:[80,85],english:100}]
```

  b. 使用JSON_VAL编写SQL

```
CREATE TABLE kafkaSource (
  `message` string
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSourceTopic>',
  'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
  'properties.group.id' = '<yourGroupId>',
  'scan.startup.mode' = 'latest-offset',
  "format" = "csv",
```

 版权所有 © 华为技术有限公司

```
  "csv.field-delimiter" = "\u0001",
  "csv.quote-character" = """
);

CREATE TABLE kafkaSink(
  message1 STRING,
  message2 STRING,
  message3 STRING,
  message4 STRING,
  message5 STRING,
  message6 STRING
) WITH (
  'connector' = 'kafka',
  'topic' = '<yourSinkTopic>',
  'properties.bootstrap.servers' =
'<yourKafkaAddress1>:<yourKafkaPort>,<yourKafkaAddress2>:<yourKafkaPort>',
  "format" = "json"
);

insert into kafkaSink select
JSON_VAL(message,""),
JSON_VAL(message,"$.name"),
JSON_VAL(message,"$.grade.science"),
JSON_VAL(message,"$.grade.science[*]"),
JSON_VAL(message,"$.grade.science[1]"),JSON_VAL(message,"$.grade.dddd")
from kafkaSource;
```

c. 查看sink中kafka的topic中的输出结果

```
{"message1":null,"message2":"swq","message3":"[80,85]","message4":"[80,85]","message5":"85"
,"message6":null}
{"message1":null,"message2":null,"message3":null,"message4":null,"message5":null,"message6":
null}
```

3.5.4.3 时间函数

Flink OpenSource SQL所支持的时间函数如表3-54所示。

函数说明

表 3-54 时间函数

函数	返回值	描述
DATE string	DATE	将日期字符串以**"yyyy-MM-dd"**的形式解析为SQL日期。
TIME string	TIME	将时间字符串以**"HH:mm:ss[.fff]"**形式解析为SQL时间。
TIMESTAMP string	TIMESTAMP	将时间字符串转换为时间戳，时间字符串格式为：**"yyyy-MM-dd HH:mm:ss[.fff]"**。

 版权所有 © 华为技术有限公司

函数	返回值	描述
INTERVAL string range	INTERVAL	interval表示时间间隔。有两种类型，分别为： • 一种为**"yyyy-MM"**即保存年份和月份，精度到月份，它的range参数可以为**YEAR**或者**YEAR To Month**。 • 一种为天时间**"dd HH:mm:sss.fff"**，用来保存天数、小时、分钟、秒和毫秒， 精度最低到毫秒。它的range参数可以为**DAY**、**MINUTE**、**DAY TO HOUR**、**DAY TO SECOND**。 例如： INTERVAL '10 00:00:00.004' DAY TO second表示间隔10天4毫秒。 INTERVAL '10' DAY表示间隔10天 INTERVAL '2-10' YEAR TO MONTH表示间隔2年10个月。
CURRENT_DATE	DATE	以UTC时区返回当前SQL日期。
CURRENT_TIME	TIME	以UTC时区返回当前SQL时间。
CURRENT_TIMESTAMP	TIMESTAMP	以UTC时区返回当前SQL时间戳。
LOCALTIME	TIME	返回当前时区的当前SQL时间。
LOCALTIMESTAMP	TIMESTAMP	返回当前时区的当前SQL时间戳。
EXTRACT(timeintervalunit FROM temporal)	BIGINT	提取时间点的一部分或者时间间隔。以int类型返回该部分。 例如：提取日期“2006-06-05”中的日为5 EXTRACT(DAY FROM DATE '2006-06-05') 返回5。
YEAR(date)	BIGINT	返回输入时间的年份 例如：YEAR(DATE '1994-09-27') 返回1994
QUARTER(date)	BIGINT	从SQL日期返回表示该日期季度的数字。
MONTH(date)	BIGINT	返回输入时间的月份 例如：MONTH(DATE '1994-09-27')返回9
WEEK(date)	BIGINT	计算当前日期是一年中的第几周 例如：WEEK(DATE '1994-09-27') 返回39
DAYOFYEAR(date)	BIGINT	计算当前日期是一年中的第几天 例如：DAYOFYEAR(DATE '1994-09-27') 返回270

版权所有 © 华为技术有限公司

函数	返回值	描述
DAYOFMONTH(date)	BIGINT	计算当前日期是这个月的第几天 例如：DAYOFMONTH(DATE '1994-09-27') 返回27
DAYOFWEEK(date)	BIGINT	计算当前日期是当前周的第几天 其中周日设为1 例如：DAYOFWEEK(DATE '1994-09-27') 返回3
HOUR(timestamp)	BIGINT	返回当前时间戳的24小时制的小时数，范围0-23 例如：HOUR(TIMESTAMP '1994-09-27 13:14:15') 返回13
MINUTE(timestamp)	BIGINT	返回当前时间戳中的分钟数，范围0-59 例如：MINUTE(TIMESTAMP '1994-09-27 13:14:15') 返回14
SECOND(timestamp)	BIGINT	返回当前时间戳中的秒数，范围0-59 例如：SECOND(TIMESTAMP '1994-09-27 13:14:15') 返回15
FLOOR(timepoint TO timeintervalunit)	TIME	向下对齐时间。 例如：FLOOR(TIME '12:44:31' TO MINUTE) 按分钟对齐到12:44:00。
CEIL(timepoint TO timeintervalunit)	TIME	向上对齐时间。 例如：CEIL(TIME '12:44:31' TO MINUTE)按分钟对齐到12:45:00。
(timepoint1, temporal1) OVERLAPS (timepoint2, temporal2)	BOOLEAN	若两个时间范围有重叠，则返回TRUE 例如： (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) 返回TRUE (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) 返回FALSE
DATE_FORMAT(timestamp, string)	STRING	将日期从源格式转换至目标格式
TIMESTAMPADD(timeintervalunit, interval, timepoint)	TIMESTAMP/ DATE/ TIME	将整型interval与timeintervalunit组成的结果添加日期或日期时间到timepoint中，并返回添加后的日期时间 例如：TIMESTAMPADD(WEEK, 1, DATE '2003-01-02') 返回2003-01-09

版权所有 © 华为技术有限公司

函数	返回值	描述
TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)	INT	返回timepoint1和timepoint2相差的时间单元数量 timepointunit表示时间单元，应该是SECOND、MINUTE、HOUR、DAY、MONTH或YEAR 例如： TIMESTAMPDIFF(DAY, TIMESTAMP '2003-01-02 10:00:00', TIMESTAMP '2003-01-03 10:00:00') 返回1
CONVERT_TZ(string1, string2, string3)	TIMESTAMP	将string2时区的时间string1转换为其在string3时区的对应时间 例如：CONVERT_TZ('1970-01-01 00:00:00', 'UTC', 'Country A/City A') 返回'1969-12-31 16:00:00'
FROM_UNIXTIME(numeric[, string])	STRING	根据时间戳numeric和当前时区返回string格式的时间，单位为秒 string默认格式为'YYYY-MM-DD hh:mm:ss' 例如：FROM_UNIXTIME(44)返回1970-01-01 09:00:44
UNIX_TIMESTAMP()	BIGINT	返回当前时间的时间戳，单位为秒
UNIX_TIMESTAMP(string1[, string2])	BIGINT	将string2格式的时间字符串string1转为时间戳，单位为秒 string2默认格式为'yyyy-MM-dd HH:mm:ss'
TO_DATE(string1[, string2])	DATE	将string2格式的日期字符串，转换为DATE类型 string2默认格式为 'yyyy-MM-dd'
TO_TIMESTAMP(string1[, string2])	TIMESTAMP	将string2格式的日期时间字符串转换为TIMESTAMP类型 string2默认格式为'yyyy-MM-dd HH:mm:ss'

DATE

- **功能描述**

  DATE函数将**"yyyy-MM-dd"**日期格式的字符串解析为DATE类型的日期。

- **语法说明**

  ```
  DATE DATE string
  ```

- **入参说明**

版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
string	STRING	SQL日期格式的字符串。 注意该字符串的格式必须为"yyyy-MM-dd"格式，否则语义校验会报错。

- **示例**
  - 测试语句

```
SELECT
    DATE "2021-08-19" AS `result`
FROM
    testtable;
```

  - 测试结果

result
2021-08-19

TIME

- **功能描述**

  将时间字符串以**"HH:mm:ss[.fff]"**形式解析为SQL时间，结果以TIME类型返回。

- **语法说明**

```
TIME TIME string
```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间字符串。 注意该字符串格式必须**"HH:mm:ss[.fff]"**，否则语义校验会报错。

- **示例**
  - 测试语句

```
SELECT
    TIME "10:11:12" AS `result`,
        TIME "10:11:12.032" AS `result2`
FROM
    testtable;
```

  - 测试结果

result	result2
10:11:12	10:11:12.032

 版权所有 © 华为技术有限公司

TIMESTAMP

- **功能描述**

  将时间字符串转换为时间戳，时间字符串格式为："**yyyy-MM-dd HH:mm:ss[.fff]**"，以TIMESTAMP(3)类型返回。

- **语法说明**

  ```
  TIMESTAMP(3) TIMESTAMP string
  ```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间戳字符串。 注意该字符串格式必须为"**yyyy-MM-dd HH:mm:ss[.fff]**"，否则语义校验会报错。

- **示例**

  - 测试语句

    ```
    SELECT
       TIMESTAMP "1997-04-25 13:14:15" AS `result`,
           TIMESTAMP "1997-04-25 13:14:15.032" AS `result2`
    FROM
       testtable;
    ```

  - 测试结果

result	result2
1997-04-25 13:14:15	1997-04-25 13:14:15.032

INTERVAL

- **功能描述**

  INTERVAL函数用于表示时间间隔。

- **语法说明**

  ```
  INTERVAL INTERVAL string range
  ```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间戳字符串，搭配参数range使用。两种格式类型，分别为： • 一种为**"yyyy-MM"**即保存年份和月份，精度到月份，它的range参数可以为**YEAR**或者**YEAR To Month**。 • 一种为天时间**"dd HH:mm:sss.fff"**，用来保存天数、小时、分钟、秒和毫秒， 精度最低到毫秒。它的range参数可以为**DAY**、**MINUTE**、**DAY TO HOUR**、**DAY TO SECOND**。

版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
range	INTERVAL	时间间隔说明，搭配string参数使用，详细请参考string参数说明。 取值范围为：**YEAR、YEAR To Month、DAY、MINUTE、DAY TO HOUR、DAY TO SECOND**。

- **示例**

  测试语句

```
--表示间隔10天4毫秒。
INTERVAL '10 00:00:00.004' DAY TO second
--DAY表示间隔10天
INTERVAL '10'
--表示间隔2年10个月
INTERVAL '2-10' YEAR TO MONTH
```

CURRENT_DATE

- **功能描述**

  以UTC时区**"yyyy-MM-dd"**格式返回当前SQL日期，返回类型为DATE。

- **语法说明**

```
DATE CURRENT_DATE
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  CURRENT_DATE AS `result`
FROM
  testtable;
```

  - 测试结果

result
2021-10-28

CURRENT_TIME

- **功能描述**

  以UTC（UTC+0）时区“HH:mm:sss.fff”格式返回当前SQL时间，返回类型为TIME。

- **语法说明**

```
TIME CURRENT_TIME
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  CURRENT_TIME AS `result`
```

 版权所有 © 华为技术有限公司

```
FROM
  testtable;
```

- 测试结果

result
08:29:19.289

CURRENT_TIMESTAMP

- **功能描述**

  以UTC（UTC+0）时区返回当前SQL时间戳，返回类型为TIMESTAMP(3)。

- **语法说明**

```
TIMESTAMP(3) CURRENT_TIMESTAMP
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  CURRENT_TIMESTAMP AS `result`
FROM
  testtable;
```

  - 测试结果

result
2021-10-28 08:33:51.606

LOCALTIME

- **功能描述**

  返回当前时区的当前SQL时间，返回类型为TIME。

- **语法说明**

```
TIME LOCALTIME
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  LOCALTIME AS `result`
FROM
  testtable;
```

  - 测试结果

result
16:39:37.706

 版权所有 © 华为技术有限公司

LOCALTIMESTAMP

- **功能描述**

  返回当前时区的当前SQL时间戳，返回类型为TIMESTAMP(3)。

- **语法说明**

```
TIMESTAMP(3) LOCALTIMESTAMP
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  LOCALTIMESTAMP AS `result`
FROM
  testtable;
```

  - 测试结果

result
2021-10-28 16:43:17.625

EXTRACT

- **功能描述**

  提取时间点或时间间隔中指定某一时间单位的部分，以BIGINT类型返回。

- **语法说明**

```
BIGINT EXTRACT(timeinteravlunit FROM temporal)
```

- **入参说明**

参数名	数据类型	参数说明
timeinteravlunit	TIMEUNIT	需要从时间点或时间间隔中提取的时间单位，取值可以是：YEAR/QUARTER/MONTH/WEEK/DAY/DOY/HOUR/MINUTE/SECOND。
temporal	DATE/TIME/TIMESTAMP/INTERVAL	时间点或时间间隔。

⚠ 注意

不允许指定不存在于时间点或时间间隔中的时间单位，否则作业会提交失败。

例如如下错误语句，会报错YEAR不能从TIME中提取。

```
SELECT
  EXTRACT(YEAR FROM TIME '12:44:31' ) AS `result`
FROM
  testtable;
```

- **示例**

版权所有 © 华为技术有限公司

- 测试语句

```
SELECT
   EXTRACT(YEAR FROM DATE '1997-04-25' ) AS `result`,
      EXTRACT(MINUTE FROM TIME '12:44:31') AS `result2`,
      EXTRACT(SECOND FROM TIMESTAMP '1997-04-25 13:14:15') AS `result3`,
      EXTRACT(YEAR FROM INTERVAL '2-10' YEAR TO MONTH) AS `result4`,
FROM
   testtable;
```

- 测试结果

result	result2	result3	result4
1997	44	15	2

YEAR

- **功能描述**

  从SQL日期date返回年份，以BIGINT类型返回。

- **语法说明**

```
BIGINT YEAR(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	DATE类型的SQL日期。

- **示例**
  - 测试语句

```
SELECT
   YEAR(DATE '1997-04-25' ) AS `result`
FROM
   testtable;
```

  - 测试结果

result
1997

QUARTER

- **功能描述**

  从SQL日期返回表示该日期季度的数字（1到4之间的整数），返回类型为BIGINT。

- **语法说明**

```
BIGINT QUARTER(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

 版权所有 © 华为技术有限公司

- **示例**
  - 测试语句
    ```
    SELECT
      QUARTER(DATE '1997-04-25' ) AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
2

MONTH

- **功能描述**

  返回输入时间的月份（1到12之间的整数），返回类型为BIGINT。
- **语法说明**
  ```
  BIGINT MONTH(date)
  ```
- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句
    ```
    SELECT
      MONTH(DATE '1997-04-25' ) AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
4

WEEK

- **功能描述**

  计算当前日期是一年中的第几周，以BIGINT类型返回。
- **语法说明**
  ```
  BIGINT WEEK(date)
  ```
- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**

 版权所有 © 华为技术有限公司

- 测试语句
```
SELECT
    WEEK(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```
- 测试结果

result
17

DAYOFYEAR

- **功能描述**

  计算当前日期是一年中的第几天（返回1到366 之间的整数），以BIGINT类型返回。
- **语法说明**
```
BIGINT DAYOFYEAR(date)
```
- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句
```
SELECT
    DAYOFYEAR(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```
  - 测试结果

result
115

DAYOFMONTH

- **功能描述**

  计算当前日期是这个月的第几天（1到31之间的整数），以BIGINT类型返回。
- **语法说明**
```
BIGINT DAYOFMONTH(date)
```
- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**

 版权所有 © 华为技术有限公司

- 测试语句

```
SELECT
    DAYOFMONTH(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

- 测试结果

result
25

DAYOFWEEK

- **功能描述**

  计算当前日期是当前周的第几天（1 到 7之间的整数），以BIGINT类型返回。

  说明

  需要注意这里自然周的起点是星期天，即每周的第1天是星期天，第2天是星期一，以此类推。

- **语法说明**

```
BIGINT DAYOFWEEK(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**

  - 测试语句

```
SELECT
    DAYOFWEEK(DATE '1997-04-25') AS `result`
FROM
    testtable;
```

  - 测试结果

result
6

HOUR

- **功能描述**

  从当前时间戳获取以24小时制的小时数进行返回，范围0-23（0 到 23 之间的整数），返回类型为BIGINT。

- **语法说明**

```
BIGINT HOUR(timestamp)
```

- **入参说明**

 版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
      HOUR(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
10

MINUTE

- **功能描述**

  返回当前时间戳中的分钟数（0 到 59 之间的整数），返回类型为BIGINT。
- **语法说明**
  ```
  BIGINT MINUTE(timestamp)
  ```
- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
      MINUTE(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
11

SECOND

- **功能描述**

  返回当前时间戳中的秒数（0 到 59 之间的整数），返回类型为BIGINT。
- **语法说明**
  ```
  BIGINT SECOND(timestamp)
  ```
- **入参说明**

 版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
       SECOND(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
       testtable;
    ```
  - 测试结果

result
12

FLOOR

- **功能描述**

  返回将时间点向下取值到指定时间单位的值。
- **语法说明**
  ```
  TIME/TIMESTAMP(3) FLOOR(timepoint TO timeintervalunit)
  ```
- **入参说明**

参数名	数据类型	参数说明
timepoint	TIMESTAMP /TIME	SQL时间或SQL时间戳。
timeintervalunit	TIMEUNIT	时间单位，类型可以是YEAR/QUARTER/ MONTH/WEEK/DAY/DOY/HOUR/MINUTE/ SECOND。

- **示例**
  - 测试语句。
    ```
    SELECT
       FLOOR(TIME '13:14:15' TO MINUTE) AS `result`
          FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
          FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`
    FROM    testtable;
    ```
  - 测试结果

message	message2	message3
13:14	13:14	1997-04-25T13:14

CEIL

- **功能描述**

  返回将时间点向上取值到指定时间单位的值。

 版权所有 © 华为技术有限公司

- **语法说明**
  TIME/TIMESTAMP(3) **CEIL**(timepoint **TO** timeintervalunit)
- **入参说明**

参数名	数据类型	参数说明
timepoint	TIMESTAMP/TIME	SQL时间或SQL时间戳。
timeintervalunit	TIMEUNIT	时间单位，类型可以是YEAR/QUARTER/MONTH/WEEK/DAY/DOY/HOUR/MINUTE/SECOND。

- **示例**
  - 测试语句。
    ```
    SELECT
        CEIL(TIME '13:14:15' TO MINUTE) AS `result`
            CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
            CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`
    FROM    testtable;
    ```
  - 测试结果

result	result2	result3
13:15	13:15	1997-04-25T13:15

OVERLAPS

- **功能描述**
  若两个时间范围有重叠，则返回TRUE，反之，则返回FALSE。
- **语法说明**
  BOOLEAN (timepoint1, temporal1) **OVERLAPS** (timepoint2, temporal2)
- **入参说明**

参数名	数据类型	参数说明
timepoint1/timepoint2	DATE/TIME/TIMESTAMP	时间点。
temporal1/temporal2	DATE/TIME/TIMESTAMP/INTERVAL	时间点或时间间隔。

 版权所有 © 华为技术有限公司

说明

- (timepoint, temporal)在判断是否重叠时为闭区间。
- temporal可以是DATE/TIME/TIMESTAMP也可以是INTERVAL。
  - 当temporal是DATE/TIME/TIMESTAMP时，(timepoint, temporal)表示timepoint, temporal之间的时间间隔。允许temporal在timepoint之前，如(DATE '1997-04-25', DATE '1997-04-23')也合法。
  - 当temporal是INTERVAL时，(timepoint, temporal)表示timepoint, timepoint +temporal之间的时间间隔。
- 必须保证(timepoint1, temporal1)和(timepoint2, temporal2)是同一数据类型的时间间隔。

- **示例**
  - 测试语句

```
SELECT
    (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS
`result`,
        (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS
`result2`,
    (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:31:00', INTERVAL '2' HOUR) AS
`result3`,
    (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:00:00', INTERVAL '3' HOUR) AS
`result4`,
    (TIMESTAMP '1997-04-25 12:00:00', TIMESTAMP '1997-04-25 12:20:00') OVERLAPS
(TIMESTAMP '1997-04-25 13:00:00', INTERVAL '2' HOUR) AS `result5`,
    (DATE '1997-04-23', INTERVAL '2' DAY) OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
AS `result6`,
    (DATE '1997-04-25', DATE '1997-04-23') OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY)
AS `result7`
FROM
    testtable;
```

  - 测试结果

result	result2	result3	result4	result5	result6	result7
true	true	false	true	false	true	true

DATE_FORMAT

- **功能描述**

  将时间戳或时间戳格式的字符串转换为指定格式的日期字符串。

- **语法说明**

```
STRING DATE_FORMAT(timestamp, dateformat)
```

- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP/ STRING	时间点。
dateformat	STRING	日期格式字符串。

 版权所有 © 华为技术有限公司

- **示例**
  - 测试语句

```
SELECT
    DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd HH:mm:ss') AS `result`,
        DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result2`,
    DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yy/MM/dd HH:mm') AS `result3`,
        DATE_FORMAT('1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result4`
FROM    testtable;
```

  - 测试结果

result	result2	result3	result4
1997-04-25 10:11:12	1997-04-25	97/04/25 10:11	1997-04-25

TIMESTAMPADD

- **功能描述**

  参考语法说明，本函数功能为将整型interval与timeintervalunit组成的结果添加到timepoint中，并返回添加后的日期时间。

  说明

  TIMESTAMPADD函数返回结果与timepoint相同。例外场景为：如果timepoint输入类型为TIMESTAMP，也可以将TIMESTAMPADD函数返回结果插入到DATE类型的表字段中。

- **语法说明**

```
TIMESTAMP(3)/DATE/TIME TIMESTAMPADD(timeintervalunit, interval, timepoint)
```

- **入参说明**

参数名	数据类型	参数说明
timeintervalunit	TIMEUNIT	时间单位。
interval	INT	整型的时间间隔。
timepoint	TIMESTAMP/DATE/TIME	时间点

- **示例**
  - 测试语句

```
SELECT
    TIMESTAMPADD(WEEK, 1, DATE '1997-04-25') AS `result`,
        TIMESTAMPADD(QUARTER, 1, TIMESTAMP '1997-04-25 10:11:12') AS `result2`,
    TIMESTAMPADD(SECOND, 2, TIME '10:11:12') AS `result3`
FROM    testtable;
```

  - 测试结果

 版权所有 © 华为技术有限公司

result	result2	result3
1997-05-02	• 如果该字段插入到TIMESTAMP类型的表字段中，则返回：1997-07-25T10:11:12 • 如果该字段插入到TIMESTAMP类型的表字段中，则返回：1997-07-25	10:11:14

TIMESTAMPDIFF

- **功能描述**

  参考语法说明，本函数功能为返回timepoint1和timepoint2之间的时间间隔，间隔的单位由第一个参数timepointunit指定。

- **语法说明**

  INT **TIMESTAMPDIFF**(timepointunit, timepoint1, timepoint2)

- **入参说明**

参数名	数据类型	参数说明
timepointunit	TIMEUNIT	时间单位。取值范围为： SECOND、MINUTE、HOUR、DAY、MONTH、YEAR。
timepoint1/timepoint2	TIMESTAMP/DATE	时间点。

- **示例**

  - 测试语句

```
SELECT
   TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-25 10:00:00', TIMESTAMP '1997-04-28 10:00:00')
AS `result`,
      TIMESTAMPDIFF(DAY, DATE '1997-04-25', DATE '1997-04-28') AS `result2`,
   TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-27 10:00:20', TIMESTAMP '1997-04-25 10:00:00')
AS `result3`
FROM    testtable;
```

  - 测试结果

result	result2	result3
3	3	-2

CONVERT_TZ

- **功能描述**

  参考语法说明，本函数将日期时间string1（具有默认ISO时间戳格式'yyyy-MM-dd HH:mm:ss'）从时区string2转换为时区string3的值，结果以STRING类型返回。

版权所有 © 华为技术有限公司

- **语法说明**
  STRING **CONVERT_TZ**(string1, string2, string3)
- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串，不符合格式的字符串会返回NULL。
string2	STRING	转换前时区。时区的格式应该是缩写如“PST”，全名如“Country A/City A”，或自定义ID如“GMT-08:00”。
string3	STRING	转换后时区。时区的格式应该是缩写如“PST”，全名如“Country A/City A”，或自定义ID如“GMT-08:00”。

- **示例**
  - 测试语句
    ```
    SELECT
       CONVERT_TZ(1970-01-01 00:00:00, UTC, Country A/City A) AS `result`,
          CONVERT_TZ(1997-04-25 10:00:00, UTC, GMT-08:00) AS `result2`
    FROM    testtable;
    ```
  - 测试结果

result	result2
1969-12-31 16:00:00	1997-04-25 02:00:00

FROM_UNIXTIME

- **功能描述**
  参考语法说明，本函数根据时间戳numeric和当前时区返回string格式的时间。
- **语法说明**
  STRING **FROM_UNIXTIME**(numeric[, string])
- **入参说明**

参数名	数据类型	参数说明
numeric	BIGINT	内部时间戳值，表示自'1970-01-01 00:00:00' UTC 以来的秒数，值可以由 UNIX_TIMESTAMP() 函数生成。
string	STRING	时间字符串格式。如果该参数不指定，则默认为'yyyy-MM-dd HH:mm:ss'。

- **示例**
  - 测试语句
    ```
    SELECT
       FROM_UNIXTIME(44) AS `result`,
          FROM_UNIXTIME(44, 'yyyy:MM:dd') AS `result2`
    FROM    testtable;
    ```

 版权所有 © 华为技术有限公司

- 测试结果

result	result2
1970-01-01 08:00:44	1970:01:01

UNIX_TIMESTAMP

- **功能描述**

  以秒为单位获取当前的Unix时间戳。以BIGINT类型返回。

- **语法说明**

```
BIGINT UNIX_TIMESTAMP()
```

- **入参说明**

  无。

- **示例**
  - 测试语句

```
SELECT
    UNIX_TIMESTAMP() AS `result`
FROM
    table;
```

  - 测试结果

result
1635401982

UNIX_TIMESTAMP(string1[, string2])

- **功能描述**

  参数语法说明，本函数将以string2格式的时间字符串string1转为Unix 时间戳（以秒为单位）。以BIGINT类型返回。

- **语法说明**

```
BIGINT UNIX_TIMESTAMP(string1[, string2])
```

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合string2参数格式的字符串语法会报错。
string2	STRING	时间字符串格式。如果不指定该参数，则默认为'yyyy-MM-dd HH:mm:ss' 。

- **示例**
  - 测试语句

```
SELECT
    UNIX_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
        UNIX_TIMESTAMP('1997-04-25 00:00:10', 'yyyy-MM-dd HH:mm:ss') AS `result2`,
        UNIX_TIMESTAMP('1997-04-25 00:00:00') AS `result3`
FROM
    testtable;
```

 版权所有 © 华为技术有限公司

- 测试结果

result	result2	result3
861897600	861897610	861897600

TO_DATE

- **功能描述**

  参数语法说明，本函数将string2格式的日期字符串string1转换为DATE类型。

- **语法说明**

```
DATE TO_DATE(string1[, string2])
```

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合格式的字符串会执行报错。
string2	STRING	字符串格式。如果不指定该参数，则默认为'yyyy-MM-dd' 。

- **示例**

  - 测试语句

```
SELECT
    TO_DATE('1997-04-25') AS `result`,
        TO_DATE('1997:04:25', 'yyyy-MM-dd') AS `result2`,
        TO_DATE('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
    testtable;
```

  - 测试结果

result	result2	result3
1997-04-25	1997-04-25	1997-04-25

TO_TIMESTAMP

- **功能描述**

  将string2格式的日期时间字符串string1转换为TIMESTAMP类型返回。

- **语法说明**

```
TIMESTAMP TO_TIMESTAMP(string1[, string2])
```

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合格式的字符串会返回NULL。

 版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
string2	STRING	日期字符串格式。如果该参数不指定，则默认为'yyyy-MM-dd HH:mm:ss' 。

- **示例**
  - 测试语句

```
SELECT
    TO_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
        TO_TIMESTAMP('1997-04-25 00:00:00') AS `result2`,
        TO_TIMESTAMP('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
    testtable;
```

  - 测试结果

result	result2	result3
1997-04-25 00:00	1997-04-25 00:00	1997-04-25 00:00

3.5.4.4 条件函数

函数说明

表 3-55 条件函数

条件函数	函数说明
CASE value WHEN value1_1 [, value1_2]* THEN result1 [WHEN value2_1 [, value2_2]* THEN result2]* [ELSE resultZ] END	当value被包含在valueX_1、valueX_2.....中时，则返回结果resultX 仅返回匹配到的第一条结果 若都不匹配，如果提供了默认值resultZ，则返回resultZ，否则返回null
CASE WHEN condition1 THEN result1 [WHEN condition2 THEN result2]* [ELSE resultZ] END	当条件表达式conditionX为TRUE时，则返回resultX 仅返回匹配到的第一条结果 若都不为TRUE，如果提供了默认值resultZ，则返回resultZ，否则返回null
NULLIF(value1, value2)	若两个值相同则返回null，否则返回value1 例如：NULLIF(5, 5)返回NULL NULLIF(5, 0)返回5

版权所有 © 华为技术有限公司

条件函数	函数说明
COALESCE(value1, value2 [, value3]*)	返回从左到右第一个不为null的参数的值 例如：COALESCE(NULL, 5)返回5
IF(condition, true_value, false_value)	若condition为TRUE则返回true_value，否则返回false_value 例如：IF(5 > 3, 5, 3)返回5
IS_ALPHA(string)	若string中的所有字符都是字母，则返回TRUE，否则返回FALSE
IS_DECIMAL(string)	若字符串可以转换为数值，则返回TRUE
IS_DIGIT(string)	若字符串中的所有字符都是数字，则返回TRUE。否则返回FALSE

3.5.4.5 类型转换函数

语法格式

```
CAST(value AS type)
```

语法说明

类型强制转换。

注意事项

- 若输入为NULL，则返回NULL。
- cast函数不支持将字符串转换为json对象类型。

示例一：将 amount 值转换成整型

将amount值转换成整型。

```
insert into temp select cast(amount as INT) from source_stream;
```

表 3-56 类型转换函数示例

示例	说明	示例
cast(v1 as string)	将v1转换为字符串类型，v1可以是数值类型，TIMESTAMP/DATE/TIME。	表T1: | content (INT)

5	 语句： SELECT cast(content as varchar) FROM T1; 结果： "5"

版权所有 © 华为技术有限公司

示例	说明	示例
cast (v1 as int)	将v1转换为int, v1可以是数值类型或字符类。	表T1: | content (STRING)

"5"	 语句: SELECT cast(content as int) FROM T1; 结果: 5
cast(v1 as timestamp)	将v1转换为timestamp类型，v1可以是字符串或DATE/TIME。	表T1: | content (STRING)

"2018-01-01 00:00:01"	 语句: SELECT cast(content as timestamp) FROM T1; 结果: 1514736001000
cast(v1 as date)	将v1转换为date类型， v1可以是字符串或者TIMESTAMP。	表T1: | content (TIMESTAMP)

1514736001000	 语句: SELECT cast(content as date) FROM T1; 结果: "2018-01-01"

说明

Flink作业不支持使用CAST将“BIGINT”转换为“TIMESTAMP”，可以使用to_timestamp进行转换。

示例二

1. 参考**Kafka源表**和**Print结果表**创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

```
CREATE TABLE kafkaSource (
  cast_int_to_string int,
  cast_String_to_int string,
  case_string_to_timestamp string,
  case_timestamp_to_date timestamp
) WITH (
  'connector' = 'kafka',
```

 版权所有 © 华为技术有限公司

```
  'topic' = '***KafkaTopic***',
  'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
  'properties.group.id' = '***GroupId***',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE printSink (
  cast_int_to_string string,
  cast_String_to_int int,
  case_string_to_timestamp timestamp,
  case_timestamp_to_date date
) WITH (
  'connector' = 'print'
);

insert into printSink select
  cast(cast_int_to_string as string),
  cast(cast_String_to_int as int),
  cast(case_string_to_timestamp as timestamp),
  cast(case_timestamp_to_date as date)
from kafkaSource;
```

2. 连接Kafka集群，向Kafka的topic中发送如下测试数据：

```
{"cast_int_to_string":"1", "cast_String_to_int": "1", "case_string_to_timestamp": "2022-04-02 15:00:00",
"case_timestamp_to_date": "2022-04-02 15:00:00"}
```

3. 查看输出结果:
   - 方法一：
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
     iii. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
   - 方法二：若在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
     iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   查询结果参考如下：

```
+I(1,1,2022-04-02T15:00,2022-04-02)
```

3.5.4.6 集合函数

函数说明

表 3-57 集合函数说明

集合函数	函数说明
CARDINALITY(array)	返回数组中元素个数
array ‘[’ integer ‘]’	返回数组索引为integer的元素。索引从1开始

 版权所有 © 华为技术有限公司

集合函数	函数说明
ELEMENT(array)	返回数组中的唯一元素。 若数组为空，则返回null 若数组中元素个数大于1，则抛出异常
CARDINALITY(map)	返回map中键值对的条数
map ‘[’ key ‘]’	返回map中key所对应的值

3.5.4.7 值构建函数

函数说明

表 3-58 值构建函数说明

值构建函数	函数说明
ROW(value1, [, value2]*) (value1, [, value2]*)	根据一系列值创建ROW
ARRAY ‘[’ value1 [, value2]* ‘]’	根据一系列值创建数组
MAP ‘[’ key1, value1 [, key2, value2]* ‘]’	根据一系列值创建MAP 其键值对为(key1, value1),(key2, value2)

3.5.4.8 属性访问函数

函数说明

表 3-59 属性访问函数说明

值接入函数	函数说明
tableName.compositeType.field	选择单个字段，通过名称访问Apache Flink复合类型（如Tuple，POJO等）的字段并返回其值。
tableName.compositeType.*	选择所有字段，将Apache Flink复合类型（如Tuple，POJO等）和其所有直接子类型转换为简单表示，其中每个子类型都是单独的字段。

版权所有 © 华为技术有限公司

3.5.4.9 Hash 函数

函数说明

表 3-60 Hash 函数说明

Hash函数	函数说明
MD5(string)	返回以32个十六进制数所表示的字符串的MD5哈希值 若字符串是null，则返回null
SHA1(string)	返回以40个十六进制所表示的字符串的SHA-1哈希值 若字符串是null，则返回null
SHA224(string)	返回以56个十六进制数所表示的字符串的SHA-224哈希值 若字符串是null，则返回null
SHA256(string)	返回以64个十六进制数所表示的字符串的SHA-256哈希值 若字符串是null，则返回null
SHA384(string)	返回以96个十六进制数所表示的字符串的SHA-384哈希值 若字符串是null，则返回null
SHA512(string)	返回以128个十六进制数所表示的字符串的SHA-512哈希值 若字符串是null，则返回null
SHA2(string, hashLength)	返回使用SHA-2哈希函数族（SHA-224, SHA-256, SHA-384, or SHA-512）得到的哈希值 第一个参数string表示被哈希的字符串，第二个参数hashLength表示哈希值的长度（224、256、384、512） 若任意参数为null，则返回null

3.5.4.10 聚合函数

聚合函数是从一组输入值计算一个结果。例如使用COUNT函数计算SQL查询语句返回的记录行数。聚合函数如表3-61所示。

表 3-61 聚合函数表

函数	返回值类型	描述
COUNT([ALL] expression | DISTINCT expression1 [, expression2]*)	BIGINT	返回表达式不为NULL的输入行数。对每个值的一个唯一实例使用DISTINCT。
COUNT(*) COUNT(1)	BIGINT	返回元组个数

版权所有 © 华为技术有限公司

函数	返回值类型	描述
AVG([ALL | DISTINCT] expression)	DOUBLE	返回所有值的平均值。 对每个值的一个唯一实例使用DISTINCT。
SUM([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值的数值之和 对每个值的一个唯一实例使用DISTINCT
MAX([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值的最大值
MIN([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值的最小值
STDDEV_POP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的总体标准偏差
STDDEV_SAMP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的样本标准偏差
VAR_POP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的总体方差
VAR_SAMP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的样本方差
COLLECT([ALL | DISTINCT] expression)	MULTISET	返回所有输入值的MULTISET
VARIANCE([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的样本方差
FIRST_VALUE(expression)	数据实际类型	返回有序数据中的第一个数据
LAST_VALUE(expression)	数据实际类型	返回有序数据中的最后一个数据

3.5.4.11 表值函数

3.5.4.11.1 string_split

string_split函数，根据指定的分隔符将目标字符串拆分为子字符串，并返回子字符串列表。

语法说明

```
string_split(target, separator)
```

 版权所有 © 华为技术有限公司

表 3-62 string_split 参数说明

参数	数据类型	说明
target	STRING	待处理的目标字符串。 **说明** • 如果target为NULL，则返回一个空行。 • 如果target包含两个或多个连续出现的分隔符时，则返回长度为零的空子字符串。 • 如果target未包含指定分隔符，则返回目标字符串。
separator	VARCHAR	指定的分隔符，当前仅支持单字符分隔。

示例

1. 参考**Kafka源表**和**Print结果表**创建flink opensource sql作业，输入以下作业运行脚本，提交运行作业。

   注意：创建作业时，在作业编辑界面的“运行参数”处，“Flink版本”选择“1.12”，勾选“保存作业日志”并设置保存作业日志的OBS桶，方便后续查看作业日志。**如下脚本中的加粗参数请根据实际环境修改**。

   ```
   CREATE TABLE kafkaSource (
     target STRING,
     separator  VARCHAR
   ) WITH (
     'connector' = 'kafka',
     'topic' = '***KafkaTopic***',
     'properties.bootstrap.servers' = '***KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort***',
     'properties.group.id' = '***GroupId***',
     'scan.startup.mode' = 'latest-offset',
     "format" = "json"
   );

   CREATE TABLE printSink (
     target STRING,
     item STRING
   ) WITH (
     'connector' = 'print'
   );

   insert into printSink
     select target,
     item from
     kafkaSource,
     lateral table(string_split(target, separator)) as T(item);
   ```

2. 连接Kafka集群，向Kafka的topic中发送如下测试数据：

   ```
   {"target":"test-flink","separator":"-"}
   {"target":"flink","separator":"-"}
   {"target":"one-two-ww-three","separator":"-"}
   ```

   即数据如下：

版权所有 © 华为技术有限公司

表 3-63 测试源表数据和分隔符

target（STRING）	separator （VARCHAR）
test-flink	-
flink	-
one-two-ww-three	-

3. 查看输出结果。
   - 方法一：
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 在对应Flink作业所在行的“操作”列，选择“更多 > FlinkUI”。
     iii. 在FlinkUI界面，选择“Task Managers”，单击对应的任务名称，选择“Stdout”查看作业运行日志。
   - 方法二：若在提交运行作业前“运行参数”选择了“保存作业日志”，可以通过如下操作查看。
     i. 登录DLI管理控制台，选择“作业管理 > Flink作业”。
     ii. 单击对应的Flink作业名称，选择“运行日志”，单击“OBS桶”，根据作业运行的日期，找到对应日志的文件夹。
     iii. 进入对应日期的文件夹后，找到名字中包含“taskmanager”的文件夹进入，下载获取taskmanager.out文件查看结果日志。

   查询结果参考如下：

```
+I(test-flink,test)
+I(test-flink,flink)
+I(flink,flink)
+I(one-two-ww-three,one)
+I(one-two-ww-three,two)
+I(one-two-ww-three,ww)
+I(one-two-ww-three,three)
```

即数据输出结果参考如下：

表 3-64 结果表数据

target（STRING）	item（STRING）
test-flink	test
test-flink	flink
flink	flink
one-two-ww-three	one
one-two-ww-three	two
one-two-ww-three	ww
one-two-ww-three	three

版权所有 © 华为技术有限公司

4 Flink Opensource SQL1.10 语法参考

4.1 SQL 语法约束与定义

4.1.1 语法支持类型

DLI SQL语法支持以下数据类型：

STRING，BOOLEAN，BYTES，DECIMAL，TINYINT，SMALLINT，INTEGER，BIGINT，FLOAT，DOUBLE，DATE，TIME，TIMESTAMP，TIMESTAMP WITH LOCAL TIME ZONE，INTERVAL，ARRAY，MULTISET，MAP，ROW

在SQL语法中这些类型用于定义表中列的数据类型。

4.1.2 语法定义

4.1.2.1 DDL 语法定义

4.1.2.1.1 CREATE TABLE 语句

语法定义

```
CREATE TABLE table_name
  (
    { <column_definition> | <computed_column_definition> }[ , ...n]
    [ <watermark_definition> ]
    [ <table_constraint> ][ , ...n]
  )
  [COMMENT table_comment]
  [PARTITIONED BY (partition_column_name1, partition_column_name2, ...)]
  WITH (key1=val1, key2=val2, ...)

<column_definition>:
  column_name column_type [ <column_constraint> ] [COMMENT column_comment]

<column_constraint>:
  [CONSTRAINT constraint_name] PRIMARY KEY NOT ENFORCED

<table_constraint>:
  [CONSTRAINT constraint_name] PRIMARY KEY (column_name, ...) NOT ENFORCED
```

 版权所有 © 华为技术有限公司

```
<computed_column_definition>:
  column_name AS computed_column_expression [COMMENT column_comment]

<watermark_definition>:
  WATERMARK FOR rowtime_column_name AS watermark_strategy_expression

<source_table>:
  [catalog_name.][db_name.]table_name
```

功能描述

根据指定的表名创建一个表。

语法说明

COMPUTED COLUMN

计算列是一个使用 “column_name AS computed_column_expression” 语法生成的虚拟列。它由使用同一表中其他列的非查询表达式生成，并且不会在表中进行物理存储。例如，一个计算列可以使用 cost AS price * quantity 进行定义，这个表达式可以包含物理列、常量、函数或变量的任意组合，但这个表达式不能存在任何子查询。

在 Flink 中计算列一般用于为 CREATE TABLE 语句定义 时间属性。 处理时间属性 可以简单地通过使用了系统函数 PROCTIME() 的 proc AS PROCTIME() 语句进行定义。另一方面，由于事件时间列可能需要从现有的字段中获得，因此计算列可用于获得事件时间列。例如，原始字段的类型不是 TIMESTAMP(3) 或嵌套在 JSON 字符串中。

注意：

- 定义在一个数据源表（ source table ）上的计算列会在从数据源读取数据后被计算，它们可以在 SELECT 查询语句中使用。
- 计算列不可以作为 INSERT 语句的目标，在 INSERT 语句中，SELECT 语句的 schema 需要与目标表不带有计算列的 schema 一致。

WATERMARK

WATERMARK 定义了表的事件时间属性，其形式为 WATERMARK FOR rowtime_column_name AS watermark_strategy_expression 。

rowtime_column_name 把一个现有的列定义为一个为表标记事件时间的属性。该列的类型必须为 TIMESTAMP(3)，且是 schema 中的顶层列，它也可以是一个计算列。

watermark_strategy_expression 定义了 watermark 的生成策略。它允许使用包括计算列在内的任意非查询表达式来计算 watermark ； 表达式的返回类型必须是 TIMESTAMP(3)，表示了从 Epoch 以来的经过的时间。 返回的 watermark 只有当其不为空且其值大于之前发出的本地 watermark 时才会被发出（以保证 watermark 递增）。每条记录的 watermark 生成表达式计算都会由框架完成。 框架会定期发出所生成的最大的 watermark ，如果当前 watermark 仍然与前一个 watermark 相同、为空、或返回的 watermark 的值小于最后一个发出的 watermark ，则新的 watermark 不会被发出。 Watermark 根据 pipeline.auto-watermark-interval 中所配置的间隔发出。 若 watermark 的间隔是 0ms ，那么每条记录都会产生一个 watermark，且 watermark 会在不为空并大于上一个发出的 watermark 时发出。

使用事件时间语义时，表必须包含事件时间属性和 watermark 策略。

Flink 提供了几种常用的 watermark 策略。

 版权所有 © 华为技术有限公司

- 严格递增时间戳： WATERMARK FOR rowtime_column AS rowtime_column。

  发出到目前为止已观察到的最大时间戳的 watermark ，时间戳大于最大时间戳的行被认为没有迟到。

- 递增时间戳： WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '0.001' SECOND。

  发出到目前为止已观察到的最大时间戳减 1 的 watermark ，时间戳大于或等于最大时间戳的行被认为没有迟到。

- 有界乱序时间戳： WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL 'string' timeUnit。

  发出到目前为止已观察到的最大时间戳减去指定延迟的 watermark ，例如，WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '5' SECOND 是一个 5 秒延迟的 watermark 策略。

```
CREATE TABLE Orders (
    user BIGINT,
    product STRING,
    order_time TIMESTAMP(3),
    WATERMARK FOR order_time AS order_time - INTERVAL '5' SECOND
) WITH ( . . . );
```

PRIMARY KEY

主键用作 Flink 优化的一种提示信息。主键限制表明一张表或视图的某个（些）列是唯一的并且不包含 Null 值。 主键声明的列都是非 nullable 的。因此主键可以被用作表行级别的唯一标识。

主键可以和列的定义一起声明，也可以独立声明为表的限制属性，不管是哪种方式，主键都不可以重复定义，否则 Flink 会报错。

有效性检查

SQL 标准主键限制可以有两种模式：ENFORCED 或者 NOT ENFORCED。 它申明了是否输入/出数据会做合法性检查（是否唯一）。Flink 不存储数据因此只支持 NOT ENFORCED 模式，即不做检查，用户需要自己保证唯一性。

Flink 假设声明了主键的列都是不包含 Null 值的，Connector 在处理数据时需要自己保证语义正确。

注意: 在 CREATE TABLE 语句中，创建主键会修改列的 nullable 属性，主键声明的列默认都是非 Nullable 的。

PARTITIONED BY

根据指定的列对已经创建的表进行分区。若表使用 filesystem sink ，则将会为每个分区创建一个目录。

WITH OPTIONS

表属性用于创建 table source/sink ，一般用于寻找和创建底层的连接器。

表达式 key1=val1 的键和值必须为字符串文本常量。

注意： 使用 CREATE TABLE 语句注册的表均可用作 table source 和 table sink。 在被 DML 语句引用前，我们无法决定其实际用于 source 抑或是 sink。

 版权所有 © 华为技术有限公司

4.1.2.1.2 CREATE VIEW 语句

语法定义

```
CREATE VIEW [IF NOT EXISTS] view_name
  [{columnName [, columnName ]* }] [COMMENT view_comment]
  AS query_expression
```

功能描述

通过定义数据视图的方式，将多层嵌套写在数据视图中，简化开发过程。

语法说明

IF NOT EXISTS

若该视图已经存在，则不会进行任何操作。

示例

创建一个名为viewName的视图

```
create view viewName as select * from dataSource
```

4.1.2.1.3 CREATE FUNCTION 语句

语法定义

```
CREATE FUNCTION
  [IF NOT EXISTS] function_name
  AS identifier [LANGUAGE JAVA|SCALA]
```

功能描述

创建一个用户自定义函数

语法说明

IF NOT EXISTS

若该函数已经存在，则不会进行任何操作。

LANGUAGE JAVA|SCALA

Language tag 用于指定 Flink runtime 如何执行这个函数。目前，只支持 JAVA 和 SCALA，且函数的默认语言为 JAVA。

示例

创建一个名为STRINGBACK的函数

```
create function STRINGBACK as 'com.dli.StringBack'
```

 版权所有 © 华为技术有限公司

4.1.2.2 DML 语法定义

DML 语句

语法定义

```
INSERT INTO table_name [PARTITION part_spec] query

part_spec:  (part_col_name1=val1 [, part_col_name2=val2, ...])

query:
  values
  | {
      select
      | selectWithoutFrom
      | query UNION [ ALL ] query
      | query EXCEPT query
      | query INTERSECT query
    }
    [ ORDER BY orderItem [, orderItem ]* ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start { ROW | ROWS } ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY]

orderItem:
  expression [ ASC | DESC ]

select:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
  [ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]

selectWithoutFrom:
  SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }

projectItem:
  expression [ [ AS ] columnAlias ]
  | tableAlias . *

tableExpression:
  tableReference [, tableReference ]*
  | tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ] JOIN tableExpression [ joinCondition ]

joinCondition:
  ON booleanExpression
  | USING '(' column [, column ]* ')'

tableReference:
  tablePrimary
  [ matchRecognize ]
  [ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ] ]

tablePrimary:
  [ TABLE ] [ [ catalogName . ] schemaName . ] tableName
  | LATERAL TABLE '(' functionName '(' expression [, expression ]* ')' ')'
  | UNNEST '(' expression ')'

values:
  VALUES expression [, expression ]*

groupItem:
  expression
  | '(' ')'
```

 版权所有 © 华为技术有限公司

```
  | '(' expression [, expression ]* ')'
  | CUBE '(' expression [, expression ]* ')'
  | ROLLUP '(' expression [, expression ]* ')'
  | GROUPING SETS '(' groupItem [, groupItem ]* ')'

windowRef:
    windowName
  | windowSpec

windowSpec:
    [ windowName ]
    '('
    [ ORDER BY orderItem [, orderItem ]* ]
    [ PARTITION BY expression [, expression ]* ]
    [
        RANGE numericOrIntervalExpression {PRECEDING}
      | ROWS numericExpression {PRECEDING}
    ]
    ')'

matchRecognize:
      MATCH_RECOGNIZE '('
      [ PARTITION BY expression [, expression ]* ]
      [ ORDER BY orderItem [, orderItem ]* ]
      [ MEASURES measureColumn [, measureColumn ]* ]
      [ ONE ROW PER MATCH ]
      [ AFTER MATCH
            ( SKIP TO NEXT ROW
            | SKIP PAST LAST ROW
            | SKIP TO FIRST variable
            | SKIP TO LAST variable
            | SKIP TO variable )
      ]
      PATTERN '(' pattern ')'
      [ WITHIN intervalLiteral ]
      DEFINE variable AS condition [, variable AS condition ]*
      ')'

measureColumn:
      expression AS alias

pattern:
      patternTerm [ '|' patternTerm ]*

patternTerm:
      patternFactor [ patternFactor ]*

patternFactor:
      variable [ patternQuantifier ]

patternQuantifier:
      '*'
  |   '*?'
  |   '+'
  |   '+?'
  |   '?'
  |   '??'
  |   '{' { [ minRepeat ], [ maxRepeat ] } '}' ['?']
  |   '{' repeat '}'
```

注意事项

Flink SQL 对于标识符（表、属性、函数名）有类似于 Java 的词法约定:

- 不管是否引用标识符，都保留标识符的大小写。
- 且标识符需区分大小写。
- 与 Java 不一样的地方在于，通过反引号，可以允许标识符带有非字母的字符（如："SELECT a AS `my field` FROM t"）。

 版权所有 © 华为技术有限公司

字符串文本常量需要被单引号包起来（如 SELECT 'Hello World' ）。两个单引号表示转移（如 SELECT 'It''s me.'）。字符串文本常量支持 Unicode 字符，如需明确使用 Unicode 编码，请使用以下语法：

- 使用反斜杠（\）作为转义字符（默认）： SELECT U&'\263A'
- 使用自定义的转义字符： SELECT U&'#263A' UESCAPE '#'

4.2 Flink OpenSource SQL1.10 语法概览

本章节介绍目前DLI所提供的Flink OpenSource SQL语法列表。参数说明，示例等详细信息请参考具体的语法说明。

创建源表相关语法

表 4-1 创建源表相关语法

语法分类	功能描述
创建源表	Kafka源表
	DIS源表
	JDBC源表
	DWS源表
	Redis源表
	Hbase源表
	userDefined源表
创建结果表	ClickHouse结果表
	Kafka结果表
	Upsert Kafka结果表
	DIS结果表
	JDBC结果表
	DWS结果表
	Redis结果表
	SMN结果表
	Hbase结果表
	Elasticsearch结果表
	userDefined结果表
创建维表	创建JDBC维表
	创建DWS维表

版权所有 © 华为技术有限公司

语法分类	功能描述
	创建Hbase维表

4.3 数据定义语句 DDL

4.3.1 创建源表

4.3.1.1 Kafka 源表

功能描述

创建source流从Kafka获取数据，作为作业的输入数据。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

前提条件

Kafka是线下集群，需要通过增强型跨源连接功能将Flink作业与Kafka进行对接。且用户可以根据实际所需设置相应安全组规则。

注意事项

对接的Kafka集群不支持开启SASL_SSL。

语法格式

```
create table kafkaSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' WATERMARK FOR rowtime_column_name AS watermark-strategy_expression)
)
with (
  'connector.type' = 'kafka',
  'connector.version' = '',
  'connector.topic' = '',
  'connector.properties.bootstrap.servers' = '',
  'connector.properties.group.id' = '',
  'connector.startup-mode' = '',
  'format.type' = ''
);
```


版权所有 © 华为技术有限公司

参数说明

表 4-2 参数说明

参数	是否必选	说明
connector.type	是	connector类型，对于kafka，需配置为'kafka'。
connector.version	是	Kafka版本，支持：'0.10'、 '0.11'。0.10或0.11版本号对应kafka版本号2.11-2.4.0及其他历史版本。
format.type	是	数据反序列化格式，支持：'csv', 'json'及'avro'等。
format.field-delimiter	否	属性分隔符，仅当编码格式为csv时，用户可以自定义属性分隔符，默认为“,”英文逗号。
connector.topic	是	kafka topic名。该参数和“connector.topic-pattern”两个参数只能使用其中一个。
connector.topic-pattern	否	匹配读取kafka topic名称的正则表达式。该参数和“connector.topic”两个参数只能使用其中一个。 例如： 'topic.*' '(topic-c|topic-d)' '(topic-a|topic-b|topic-\\d*)' '(topic-a|topic-b|topic-[0-9]*)'
connector.properties.bootstrap.servers	是	kafka brokers地址，以逗号分隔。
connector.properties.group.id	否	消费组名称
connector.startup-mode	否	consumer启动模式，支持：'earliest-offset', 'latest-offset', 'group-offsets', 'specific-offsets'及'timestamp'。默认值为'group-offsets'。
connector.specific-offsets	否	指定消费offset，'startup-mode'为'specific-offsets'时需配置，格式为: 'partition:0,offset:42;partition:1,offset:300'。
connector.startup-timestamp-millis	否	指定起始消费时间戳，'startup-mode'为'timestamp'时需配置。
connector.properties.*	否	配置kafka任意原生属性。

版权所有 © 华为技术有限公司

示例

- 从Kafka中读取编码格式为csv，对象为kafkaSource的表。

```
create table kafkaSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test-topic',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.properties.group.id' = 'test-group',
  'connector.startup-mode' = 'latest-offset',
  'format.type' = 'csv'
);
```

- 从Kafka中读取编码格式为不含嵌套的json数据，对象为kafkaSource的表。

  例如不含嵌套的json数据格式为：

```
{"car_id": 312, "car_owner": "wang", "car_brand": "tang"}
{"car_id": 313, "car_owner": "li", "car_brand": "lin"}
{"car_id": 314, "car_owner": "zhao", "car_brand": "han"}
```

  则创建表语句为：

```
create table kafkaSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING
)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test-topic',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.properties.group.id' = 'test-group',
  'connector.startup-mode' = 'latest-offset',
  'format.type' = 'json'
);
```

- 从Kafka中读取编码格式包含嵌套的json数据，对象为kafkaSource的表。

  例如包含嵌套的json数据格式为：

```
{
    "id":"1",
    "type":"online",
    "data":{
        "patient_id":1234,
        "name":"bob1234",
        "age":"Bob",
        "gmt_create":"Bob",
        "gmt_modify":"Bob"
    }
}
```

  则创建表语句为：

```
CREATE table kafkaSource(
  id STRING,
  type STRING,
  data ROW(
    patient_id STRING,
    name STRING,
    age STRING,
    gmt_create STRING,
    gmt_modify STRING)
)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test-topic',
```

 版权所有 © 华为技术有限公司

```
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.properties.group.id' = 'test-group',
  'connector.startup-mode' = 'latest-offset',
  'format.type' = 'json'
);
```

4.3.1.2 DIS 源表

功能描述

创建source流从数据接入服务（DIS）获取数据。用户数据从DIS接入，Flink作业从DIS的通道读取数据，作为作业的输入数据。Flink作业可通过DIS的source源将数据从生产者快速移出，进行持续处理，适用于将云服务外数据导入云服务后进行过滤、实时分析、监控报告和转储等场景。

数据接入服务（Data Ingestion Service，简称DIS）为处理或分析流数据的自定义应用程序构建数据流管道，主要解决云服务外的数据实时传输到云服务内的问题。数据接入服务每小时可从数十万种数据源（如IoT数据采集、日志和定位追踪事件、网站点击流、社交媒体源等）中连续捕获、传送和存储数TB数据。DIS的更多信息，请参见**《数据接入服务用户指南》**。

语法格式

```
create table disSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'dis',
  'connector.region' = '',
  'connector.channel' = '',
  'format-type' = ''
);
```

参数说明

表 4-3 参数说明

参数	是否必选	说明
connector.type	是	数据源类型，“dis”表示数据源为数据接入服务，必须为dis。
connector.region	是	数据所在的DIS区域。
connector.ak	否	访问密钥ID(Access Key ID)，需与sk同时设置
connector.sk	否	Secret Access Key，需与ak同时设置
connector.channel	是	数据所在的DIS通道名称。

 版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.partition-count	否	读取从0分区开始计算的partition-count个通道范围内的数据。 该参数和partition-range参数不能同时配置。 当两个参数都没有配置的时候默认读取所有partition。
connector.partition-range	否	指定作业从DIS通道读取的分区范围。该参数和partition-count参数不能同时配置。当两个参数没有配置的时候默认读取所有partition。 partition-range = "[0:2]"时，表示读取的分区范围是1-3，包括分区1、分区2和分区3，范围设置要在dis相应通道的范围内。
connector.offset	否	用户可以根据需求设置该参数的数值，读取数据的起始位置，与start-time不能同时设置。
connector.start-time	否	DIS数据读取从该起始时间的数据。 当该参数配置时则从配置的时间开始读取数据，有效格式为yyyy-MM-dd HH:mm:ss。 当没有配置start-time也没配置offset的时候，读取最新数据。
connector. enable-checkpoint	否	是否启用checkpoint功能，可配置为true（启用）或者false（停用）， 默认为false。 勿与offset或start-time同时设置；若enable-checkpoint为true，与checkpoint-app-name需要同时配置。
connector. checkpoint-app-name	否	DIS服务的消费者标识，当不同作业消费相同通道时，需要区分不同的消费者标识，以免checkpoint混淆。 勿与offset或start-time同时设置；若enable-checkpoint为true，则需要同时配置。
connector. checkpoint-interval	否	DIS源算子做checkpoint的时间间隔，默认为60s。格式为d、day/h、hour/min、minute/s、sec、second 勿与offset或start-time同时设置。
format.type	是	数据编码格式，可选为“csv”、“json”
format.field-delimiter	否	属性分隔符，仅当编码格式为csv时，用户可以自定义属性分隔符，默认为“,”英文逗号。

注意事项

无

版权所有 © 华为技术有限公司

示例

```
create table disCsvSource (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT)
with (
  'connector.type' = 'dis',
  'connector.region' = 'cn-north-1',
  'connector.channel' = 'disInput',
  'format.type' = 'csv'
);
```

4.3.1.3 JDBC 源表

功能描述

JDBC连接器是Flink内置的Connector，用于从数据库读取相应的数据。

前提条件

- 要与实例建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

语法格式

```
create table jbdcSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'jdbc',
  'connector.url' = '',
  'connector.table' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

参数说明

表 4-4 参数说明

参数	是否必选	说明
connector.type	是	数据源类型，‘jdbc’表示使用JDBC connector，必须为jdbc
connector.url	是	数据库的URL
connector.table	是	读取数据库中的数据所在的表名

 版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.driver	否	连接数据库所需要的驱动。若未配置，则会自动通过URL提取
connector.username	否	数据库认证用户名，需要和'connector.password'一起配置
connector.password	否	数据库认证密码，需要和'connector.username'一起配置
connector.read.partition.column	否	用于对输入进行分区的列名 与connector.read.partition.lower-bound、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.lower-bound	否	第一个分区的最小值 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.upper-bound	否	最后一个分区的最大值 与connector.read.partition.column、connector.read.partition.lower-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.num	否	分区的个数 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.upper-bound必须同时存在或者同时不存在
connector.read.fetch-size	否	每次从数据库拉取数据的行数。默认值为0，表示忽略该提示。

注意事项

无

示例

```
create table jdbcSource (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
```


版权所有 © 华为技术有限公司

```
  total_miles INT)
with (
  'connector.type' = 'jdbc',
  'connector.url' = 'jdbc:mysql://xx.xx.xx.xx:3306/xx',
  'connector.table' = 'jdbc_table_name',
  'connector.driver' = 'com.mysql.jdbc.Driver',
  'connector.username' = 'xxx',
  'connector.password' = 'xxxxxx'
);
```

4.3.1.4 DWS 源表

功能描述

DLI将Flink作业从数据仓库服务（DWS）中读取数据。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见**《数据仓库服务管理指南》**。

前提条件

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。
  如何创建DWS集群，请参考《数据仓库服务管理指南》中“创建集群”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
- 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'gaussdb',
  'connector.url' = '',
  'connector.table' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

 版权所有 © 华为技术有限公司

参数说明

表 4-5 参数说明

参数	是否必选	说明
connector.type	是	connector类型，需配置为'gaussdb'
connector.url	是	jdbc连接地址，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。DWS数据库版本为8.1.0以后的版本时，格式为：jdbc:gaussdb://${ip}:${port}/${dbName}。
connector.table	是	操作的表名。如果该DWS表在某schema下，则格式为：'schema\".\"具体表名'，具体可以参考**示例**说明。
connector.driver	否	jdbc连接驱动，默认为: org.postgresql.Driver。 DWS数据库版本为8.1.0以后的版本时，连接驱动为：com.huawei.gauss200.jdbc.Driver。
connector.username	否	数据库认证用户名，需要和'connector.password'一起配置
connector.password	否	数据库认证密码，需要和'connector.username'一起配置
connector.read.partition.column	否	用于对输入进行分区的列名 与connector.read.partition.lower-bound、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.lower-bound	否	第一个分区的最小值 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.upper-bound	否	最后一个分区的最大值 与connector.read.partition.column、connector.read.partition.lower-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.num	否	分区的个数 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.upper-bound必须同时存在或者同时不存在

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.read.fetch-size	否	每次从数据库拉取数据的行数。默认值为0，表示忽略该提示

示例

- 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。该驱动为默认，创建表时可以不填该驱动参数。

  表car_info没有在schema下时。

```
create table dwsSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'gaussdb',
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/xx',
  'connector.table' = 'car_info',
  'connector.username' = 'xx',
  'connector.password' = 'xx'
);
```

  当DWS表test在名为test_schema的schema下时，可以参考如下样例。

```
create table dwsSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'gaussdb',
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/xx',
  'connector.table' = 'test_schema\".\"test',
  'connector.username' = 'xx',
  'connector.password' = 'xx'
);
```

- 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。

  当DWS表test在名为ads_game_sdk_base的schema下时，可以参考如下样例。

```
create table dwsSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'gaussdb',
  'connector.table' = 'ads_game_sdk_base\".\"test',
  'connector.driver' = 'com.huawei.gauss200.jdbc.Driver',
  'connector.url' = 'jdbc:gaussdb://xx.xx.xx.xx:8000/xx',
  'connector.username' = 'xx',
  'connector.password' = 'xx'
);
```

4.3.1.5 Redis 源表

功能描述

创建source流从Redis获取数据，作为作业的输入数据。

版权所有 © 华为技术有限公司

前提条件

要建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'redis',
  'connector.host' = '',
  'connector.port' = ''
);
```

参数说明

表 4-6 参数说明

参数	是否必选	说明
connector.type	是	connector类型，对于redis，需配置为'redis'。
connector.host	是	redis连接地址。
connector.port	是	redis连接端口。
connector.password	否	redis认证密码。
connector.deploy-mode	否	redis部署模式，支持standalone/cluster，默认standalone。
connector.table-name	否	table存储模式下必配，redis中存储表名。在table存储模式下，数据将以hash类型存储到redis，其中key为：${table-name}:${ext-key}，field名为列名。 **说明** table存储模式：将connector.table-name、connector.key-column作为redis的key。redis的hash类型，每个key对应一个hashmap，hashmap的hashkey为源表的字段名，hashvalue为源表的字段值。
connector.use-internal-schema	否	table存储模式下可配置，是否使用redis中已存在schema，默认为false。
connector.key-column	否	table存储模式下可配置，将该字段值作为redis中的ext-key，未配置时，ext-key为生成的uuid。

版权所有 © 华为技术有限公司

示例

从Redis中读取数据。

```
create table redisSource(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
 'connector.type' = 'redis',
  'connector.host' = 'xx.xx.xx.xx',
  'connector.port' = '6379',
  'connector.password' = 'xx',
  'connector.table-name' = 'car_info'
);
```

4.3.1.6 Hbase 源表

功能描述

创建source流从HBase中获取数据，作为作业的输入数据。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。DLI可以从HBase中读取数据，用于过滤分析、数据转储等场景。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
- **若使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《 数据湖探索用户指南 》中的“**修改主机信息**”章节描述。
- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

语法格式

```
create table hbaseSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = '',
  'connector.zookeeper.quorum' = ''
);
```


版权所有 © 华为技术有限公司

参数说明

表 4-7 参数说明

参数	是否必选	说明
connector.type	是	connector的类型，只能为hbase
connector.version	是	该值只能为1.4.3
connector.table-name	是	hbase中的表名
connector.zookeeper.quorum	是	Zookeeper的地址
connector.zookeeper.znode.parent	否	Zookeeper中的根目录，默认是/hbase
connector.rowkey	否	读取复合rowkey的内容，并根据设置的大小，赋予给新的字段 形如：rowkey1:3,rowkey2:3,… 其中3表示取该字段的前3个byte，该值不能大于该字段的字节大小，且该值不能小于1。表示将复合rowkey的前三个字节赋给字段rowkey1，其后三个字节赋给字段rowkey2

示例

```
create table hbaseSource(
  rowkey1 string,
  rowkey2 string,
  info Row<owner string>,
  car ROW<miles string, speed string>
 ) with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = 'carinfo',
  'connector.rowkey' = 'rowkey1:1,rowkey2:3',
  'connector.zookeeper.quorum' = 'xxxx:2181'
 );
```

4.3.1.7 userDefined 源表

功能描述

您可通过编写代码实现从云生态或者开源生态获取数据，再把获取到的数据作为Flink作业的输入数据。

前提条件

自定义source类需要继承类RichParallelSourceFunction，并指定数据类型为Row。

版权所有 © 华为技术有限公司

例如自定义类MySource：**public class MySource extends RichParallelSourceFunction<Row>{}**，重点实现其中的open、run、close和cancel函数。实现完成后将该类编译打在jar中，通过sql编辑页的UDF Jar上传。

依赖的pom配置文件内容参考如下：

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java_2.11</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-core</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
```

语法格式

```
create table userDefinedSource (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = ''
);
```

参数说明

表 4-8 参数说明

参数	是否必选	说明
connector.type	是	只能为user-defined，表示使用自定义的source。
connector.class-name	是	source函数的全限定类名。
connector.class-parameter	否	source函数其构造函数的参数，只支持一个String类型的参数。

注意事项

connector.class-name需要为全限定类名。

示例

```
create table userDefinedSource (
  attr1 int,
  attr2 int
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = 'xx.xx.MySource'
);
```

 版权所有 © 华为技术有限公司

4.3.2 创建结果表

4.3.2.1 ClickHouse 结果表

功能描述

DLI将Flink作业数据输出到ClickHouse中。

ClickHouse是面向联机分析处理的列式数据库，支持SQL查询，且查询性能好，特别是基于大宽表的聚合分析查询性能非常优异，比其他分析型数据库速度快一个数量级。详细请参考**ClickHouse组件操作**。

前提条件

该场景需要与ClickHouse建立增强型跨源连接，并根据实际情况设置ClickHouse集群所在安全组规则中的端口。

建立增强型跨源连接，请参考《 数据湖探索用户指南 》中的“增强型跨源连接”章节。

如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

注意事项

- 创建MRS的ClickHouse集群，集群版本选择MRS 3.1.0，且勿开启kerberos认证。
- Flink SQL语句中不能定义主键。同时不能使用任何产生主键的语法，例如insert into clickhouseSink select id, cout(*) from sourceName group by id。
- Flink中支持字段类型范围为：string、tinyint、smallint、int、long、float、double、date、timestamp、decimal以及Array。

  其中Array中的数据类型仅支持int、bigint、string、float、double。

语法格式

```
create table clickhouseSink (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = 'clickhouse',
  'connector.url' = '',
  'connector.table' = ''
);
```

参数说明

表 4-9 参数说明

参数	是否必选	说明
connector.type	是	固定为clickhouse

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.url	是	ClickHouse的url。 参数格式为：**jdbc:clickhouse://**ClickHouseBalancer实例的IP:ClickHouseBalancer实例的http端口/数据库名 ● ClickHouseBalancer实例的IP地址：登录MRS管理控制台，选择“集群名称 > 组件管理 > ClickHouse > 实例”，获取ClickHouseBalancer实例的业务IP。 ● ClickHouseBalancer实例的http端口：登录MRS管理控制台，选择“集群名称 > 组件管理 > ClickHouse > 服务配置”，角色选择“ClickHouseBalancer”，搜索“lb_http_port”配置参数值。默认为：21425。 ● 数据库名为ClickHouse集群创建的数据库名称。
connector.table	是	要创建的ClickHouse的表名。
connector.driver	否	连接数据库所需要的驱动。 ● 如果建表时不指定该参数，驱动会自动通过ClickHouse的url提取。 ● 如果建表时指定该参数，则该参数值固定为“ru.yandex.clickhouse.ClickHouseDriver”。
connector.username	否	访问ClickHouse数据库的账号。
connector.password	否	访问ClickHouse数据库账号的密码。
connector.write.flush.max-rows	否	写数据时刷新数据的最大行数，默认值为：5000。
connector.write.flush.interval	否	刷新数据的时间间隔，单位可以为ms、milli、millisecond/s、sec、second/min、minute等。
connector.write.max-retries	否	写数据失败时的最大尝试次数，默认值为：3。

示例

从dis中读取数据，并将数据插入到数据库为flinktest、表名为test的ClickHouse数据库中。

1. 创建dis数据源表disSource。

```
create table disSource(
  attr0 string,
  attr1 TINYINT,
  attr2 smallint,
  attr3 int,
  attr4 bigint,
```

 版权所有 © 华为技术有限公司

```
  attr5 float,
  attr6 double,
  attr7 String,
  attr8 string,
  attr9 timestamp(3),
  attr10 timestamp(3),
  attr11 date,
  attr12 decimal(38, 18),
  attr13 decimal(38, 18)
) with (
  "connector.type" = "dis",
  "connector.region" = "cn-xxxx-x",
  "connector.channel" = "xxxx",
  "format.type" = 'csv'
);
```

2. 创建ClickHouse结果表clickhouse，将disSource表数据插入到clickhouse结果表中。

```
create table clickhouse(
  attr0 string,
  attr1 TINYINT,
  attr2 smallint,
  attr3 int,
  attr4 bigint,
  attr5 float,
  attr6 double,
  attr7 String,
  attr8 string,
  attr9 timestamp(3),
  attr10 timestamp(3),
  attr11 date,
  attr12 decimal(38, 18),
  attr13 decimal(38, 18),
  attr14 array < int >,
  attr15 array < bigint >,
  attr16 array < float >,
  attr17 array < double >,
  attr18 array < varchar >,
  attr19 array < String >
) with (
  'connector.type' = 'clickhouse',
  'connector.url' = 'jdbc:clickhouse://xx.xx.xx.xx:xx/flinktest',
  'connector.table' = 'test'
);

insert into
  clickhouse
select
  attr0,
  attr1,
  attr2,
  attr3,
  attr4,
  attr5,
  attr6,
  attr7,
  attr8,
  attr9,
  attr10,
  attr11,
  attr12,
  attr13,
  array [attr3, attr3+1],
  array [cast(attr4 as bigint), cast(attr4+1 as bigint)],
  array [cast(attr12 as float), cast(attr12+1 as float)],
  array [cast(attr13 as double), cast(attr13+1 as double)],
  array ['TEST1', 'TEST2'],
  array [attr7, attr7]
from
  disSource;
```

 版权所有 © 华为技术有限公司

4.3.2.2 Kafka 结果表

功能描述

DLI将Flink作业的输出数据输出到Kafka中。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

前提条件

Kafka是线下集群，需要通过增强型跨源连接功能将Flink作业与Kafka进行对接。且用户可以根据实际所需设置相应安全组规则。

注意事项

对接的Kafka集群不支持开启SASL_SSL。

语法格式

```
create table kafkaSource(
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'kafka',
  'connector.version' = '',
  'connector.topic' = '',
  'connector.properties.bootstrap.servers' = '',
  'format.type' = ''
);
```

参数说明

表 4-10 参数说明

参数	是否必选	说明
connector.type	是	connector类型，对于kafka，需配置为'kafka'。
connector.version	否	Kafka版本，支持：'0.10'、 '0.11'。0.10或0.11版本号对应kafka版本号2.11-2.4.0及其他历史版本。
format.type	是	数据序列化格式，支持：'csv'、 'json'及'avro'等。
format.field-delimiter	否	属性分隔符，仅当编码格式为csv时，用户可以自定义属性分隔符，默认为“,”英文逗号。
connector.topic	是	kafka topic名。
connector.properties.bootstrap.servers	是	kafka brokers地址，以逗号分隔。

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.sink-partitioner	否	记录分区的方式，支持：'fixed', 'round-robin'及'custom'。
connector.sink-partitioner-class	否	'sink-partitioner'为'custom'时，需配置，如'org.mycompany.MyPartitioner' 。
update-mode	否	支持：'append'、'retract'及'upsert'三种写入模式。
connector.properties.*	否	配置kafka任意原生属性

示例

将kafkaSink的数据输出到Kafka中

```
create table kafkaSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.10',
  'connector.topic' = 'test-topic',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.sink-partitioner' = 'round-robin',
  'format.type' = 'csv'
);
```

4.3.2.3 Upsert Kafka 结果表

功能描述

DLI将Flink作业的输出数据以upsert的模式输出到Kafka中。

Apache Kafka是一个快速、可扩展的、高吞吐、可容错的分布式发布订阅消息系统，具有高吞吐量、内置分区、支持数据副本和容错的特性，适合在大规模消息处理场景中使用。

前提条件

Kafka是线下集群，需要通过增强型跨源连接功能将Flink作业与Kafka进行对接。且用户可以根据实际所需设置相应安全组规则。

注意事项

对接的Kafka集群不支持开启SASL_SSL。

语法格式

```
create table kafkaSource(
  attr_name attr_type
  (',' attr_name attr_type)*
```

 版权所有 © 华为技术有限公司

```
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'upsert-kafka',
  'connector.version' = '',
  'connector.topic' = '',
  'connector.properties.bootstrap.servers' = '',
   'format.type' = ''
);
```

参数说明

表 4-11 参数说明

参数	是否必选	说明
connector.type	是	connector类型，对于upsert kafka，需配置为'upsert-kafka'
connector.version	否	Kafka版本，仅支持：'0.11'
format.type	是	数据序列化格式，支持：'csv', 'json'及'avro'等
connector.topic	是	kafka topic名
connector.properties.bootstrap.servers	是	kafka brokers地址，以逗号分隔
connector.sink-partitioner	否	记录分区方式，支持：'fixed', 'round-robin'及'custom'
connector.sink-partitioner-class	否	'sink-partitioner'为'custom'时，需配置，如'org.mycompany.MyPartitioner'
connector.sink.ignore-retraction	否	是否忽略回撤消息，默认为false。回撤消息将以null值写入kafka
update-mode	否	支持：'append', 'retract'及'upsert'三种写入模式
connector.properties.*	否	配置kafka任意原生属性

 版权所有 © 华为技术有限公司

示例

```
create table upsertKafkaSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT,
  primary key (car_id) not enforced
)
with (
  'connector.type' = 'upsert-kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test-topic',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'format.type' = 'csv'
);
```

4.3.2.4 DIS 结果表

功能描述

DLI将Flink作业的输出数据写入数据接入服务（DIS）中。适用于将数据过滤后导入DIS通道，进行后续处理的场景。

数据接入服务（Data Ingestion Service，简称DIS）为处理或分析流数据的自定义应用程序构建数据流管道，主要解决云服务外的数据实时传输到云服务内的问题。数据接入服务每小时可从数十万种数据源（如IoT数据采集、日志和定位追踪事件、网站点击流、社交媒体源等）中连续捕获、传送和存储数TB数据。DIS的更多信息，请参见《数据接入服务用户指南》。

语法格式

```
create table disSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'dis',
  'connector.region' = '',
  'connector.channel' = '',
  'format.type' = ''
);
```

参数说明

表 4-12 参数说明

参数	是否必选	说明
connector.type	是	数据源类型，“dis”表示数据源为数据接入服务，必须为dis。
connector.region	是	数据所在的DIS区域。
connector.ak	否	访问密钥ID(Access Key ID)，需与sk同时设置
connector.sk	否	Secret Access Key，需与ak同时设置

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.channel	是	数据所在的DIS通道名称。
format.type	是	数据编码格式，可选为“csv”、“json”
format.field-delimiter	否	属性分隔符，仅当编码格式为csv时，用户可以自定义属性分隔符，默认为“,”英文逗号。
connector.partition-key	否	数据输出分组主键，多个主键用逗号分隔。当该参数没有配置的时候则随机派发。

注意事项

无

示例

将流disSink的数据输出到DIS中。

```
create table disSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
)
with (
  'connector.type' = 'dis',
  'connector.region' = 'cn-north-1',
  'connector.channel' = 'disOutput',
  'connector.partition-key' = 'car_id,car_owner',
  'format.type' = 'csv'
);
```

4.3.2.5 JDBC 结果表

功能描述

DLI将Flink作业的输出数据输出到关系型数据库中。

前提条件

- 要与实例建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
- 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

语法格式

```
create table jdbcSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
```


版权所有 © 华为技术有限公司

```
with (
  'connector.type' = 'jdbc',
  'connector.url' = '',
  'connector.table' = '',
  'connector.driver' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

参数说明

表 4-13 参数说明

参数	是否必选	说明
connector.type	是	数据源类型，‘jdbc’表示使用JDBC connector，必须为jdbc
connector.url	是	数据库的URL
connector.table	是	读取数据库中的数据所在的表名
connector.driver	否	连接数据库所需要的驱动。若未配置，则会自动通过URL提取
connector.username	否	访问数据库所需要的账号
connector.password	否	访问数据库所需要的密码
connector.write.flush.max-rows	否	写数据时，刷新数据的最大行数。默认值为5000
connector.write.flush.interval	否	刷新数据的时间间隔，单位可以为ms、milli、millisecond/s、sec、second/min、minute等。不填写则默认不根据时间刷新
connector.write.max-retries	否	写数据失败时的最大尝试次数。默认值为3
connector.write.exclude-update-columns	否	默认值为空（默认忽略primary key字段），表示更新主键值相同的数据时，忽略指定字段的更新

注意事项

无

示例

将流jdbcSink的数据输出到MySQL数据库中。

```
create table jdbcSink(
  car_id STRING,
```

 版权所有 © 华为技术有限公司

```
  car_owner STRING,
  car_brand STRING,
  car_speed INT
)
with (
  'connector.type' = 'jdbc',
  'connector.url' = 'jdbc:mysql://xx.xx.xx.xx:3306/xx',
  'connector.table' = 'jdbc_table_name',
  'connector.driver' = 'com.mysql.jdbc.Driver',
  'connector.username' = 'xxx',
  'connector.password' = 'xxxxxx'
);
```

4.3.2.6 DWS 结果表

功能描述

DLI将Flink作业的输出数据输出到数据仓库服务（DWS）中。DWS数据库内核兼容PostgreSQL，PostgreSQL数据库可存储更加复杂类型的数据，支持空间信息服务、多版本并发控制（MVCC）、高并发，适用场景包括位置应用、金融保险、互联网电商等。

数据仓库服务（Data Warehouse Service，简称DWS）是一种基于基础架构和平台的在线数据处理数据库，为用户提供海量数据挖掘和分析服务。DWS的更多信息，请参见**《数据仓库服务管理指南》**。

前提条件

- 请务必确保您的账户下已在数据仓库服务（DWS）里创建了DWS集群。

  如何创建DWS集群，请参考《数据仓库服务管理指南》中“创建集群”章节。
- 请确保已创建DWS数据库表。
- 该场景作业需要运行在DLI的独享队列上，因此要与DWS集群建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
- 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

语法格式

说明

DWS结果表中不允许指定所有属性为PRIMARY KEY。

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'gaussdb',
  'connector.url' = '',
  'connector.table' = '',
  'connector.driver' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

 版权所有 © 华为技术有限公司

参数说明

表 4-14 参数说明

参数	是否必选	说明
connector.type	是	connector类型，需配置为'gaussdb'
connector.url	是	jdbc连接地址，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。
connector.table	是	操作的表名。如果该DWS表在某schema下，则格式为：'schema\".\"具体表名'，具体可以参考**示例**说明。
connector.driver	否	jdbc连接驱动，默认为: org.postgresql.Driver。
connector.username	否	数据库认证用户名，需要和'connector.password'一起配置
connector.password	否	数据库认证密码，需要和'connector.username'一起配置
connector.write.mode	否	数据写入模式，支持: copy, insert以及upsert三种。默认值为upsert。 该参数与'primary key'配合使用。 • 未配置'primary key'时，支持copy及insert两种模式追加写入。 • 配置'primary key'，支持copy、upsert以及insert三种模式更新写入。 注意：由于dws不支持更新分布列，因而配置的更新主键必须包含dws表中定义的所有分布列。
connector.write.flush.max-rows	否	数据flush大小，超过该值将触发写入flush。默认为5000。
connector.write.flush.interval	否	数据flush周期，周期性触发写入flush。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。不填写则默认不根据时间刷新。
connector.write.max-retries	否	写入最大重试次数，默认为3。
connector.write.merge.filter-key	否	配置PRIMARY KEY，并且“connector.write.mode”配置为copy时，可以配置merge时的过滤列名。
connector.write.escape-string-value	否	是否对string类型值进行转义，默认为false。

版权所有 © 华为技术有限公司

注意事项

无

示例

- 使用gsjdbc4驱动连接时，加载的数据库驱动类为：org.postgresql.Driver。该驱动为默认，创建表时可以不填该驱动参数。
  - 使用upsert模式，写入数据到DWS

```
create table dwsSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'gaussdb',
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/xx',
  'connector.table' = 'car_info',
  'connector.username' = 'xx',
  'connector.password' = 'xx',
  'connector.write.mode' = 'upsert',
  'connector.write.flush.interval' = '30s'
);
```

    当DWS表test在名为ads_game_sdk_base的schema下时，可以参考如下样例。

```
CREATE TABLE ads_rpt_game_sdk_realtime_ada_reg_user_pay_mm (
  ddate DATE,
  dmin TIMESTAMP(3),
  game_appkey VARCHAR,
  channel_id VARCHAR,
  pay_user_num_1m bigint,
  pay_amt_1m bigint,
  PRIMARY KEY (ddate, dmin, game_appkey, channel_id) NOT ENFORCED
) WITH (
  'connector.type' = 'gaussdb',
  'connector.url' = 'jdbc:postgresql://xx.xx.xx.xx:8000/dws_bigdata_db',
  'connector.table' = 'ads_game_sdk_base\".\"test',
  'connector.username' = 'xxxx',
  'connector.password' = 'xxxxx',
  'connector.write.mode' = 'upsert',
  'connector.write.flush.interval' = '30s'
);
```

- 使用gsjdbc200驱动连接时，加载的数据库驱动类为：com.huawei.gauss200.jdbc.Driver。

  当DWS表test在名为ads_game_sdk_base的schema下时，可以参考如下样例。

```
create table dwsSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'gaussdb',
  'connector.table' = 'ads_game_sdk_base\".\"test',
  'connector.driver' = 'com.huawei.gauss200.jdbc.Driver',
  'connector.url' = 'jdbc:gaussdb://xx.xx.xx.xx:8000/xx',
  'connector.username' = 'xx',
  'connector.password' = 'xx',
  'connector.write.mode' = 'upsert',
  'connector.write.flush.interval' = '30s'
);
```

 版权所有 © 华为技术有限公司

4.3.2.7 Redis 结果表

功能描述

DLI将Flink作业的输出数据输出到Redis中。Redis是一种支持Key-Value等多种数据结构的存储系统。可用于缓存、事件发布或订阅、高速队列等场景，提供字符串、哈希、列表、队列、集合结构直接存取，基于内存，可持久化。有关Redis的详细信息，请访问Redis官方网站**https://redis.io/**。

前提条件

要建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

语法格式

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
'connector.type' = 'redis',
  'connector.host' = '',
  'connector.port' = '',
  'connector.password' = '',
  'connector.table-name' = '',
  'connector.key-column' = ''
);
```

参数说明

表 4-15 参数说明

参数	是否必选	说明
connector.type	是	connector类型，对于redis，需配置为'redis'。
connector.host	是	redis连接地址。
connector.port	是	redis连接端口。
connector.password	否	redis认证密码。
connector.deploy-mode	否	redis部署模式，支持standalone/cluster，默认standalone

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.table-name	否	table存储模式下必配，redis中存储表名。在table存储模式下，数据将以hash类型存储到redis，其中key为：${table-name}:${ext-key}，field名为列名。 **说明** table存储模式：将connector.table-name、connector.key-column作为redis的key。redis的hash类型，每个key对应一个hashmap，hashmap的hashkey为源表的字段名，hashvalue为源表的字段值。
connector.key-column	否	table存储模式下可配置，将该字段值作为redis中的ext-key，未配置时，ext-key为生成的uuid
connector.write-schema	否	table存储模式下可配置，是否将当前schema写入到redis，默认为false
connector.data-type	否	数据存储类型，用户自定义存储模式必配。支持：string, list, hash, set类型。其中string/list以及sets中schema字段数必须为2，hash字段数必须为3
connector.ignore-retraction	否	是否忽略retraction消息，默认为false

注意事项

参数“connector.table-name”与“connector.data-type”必须配置其中一个。

示例

- 配置“connector.table-name”参数时的table存储模式示例。

  table模式采用hash类型存储数据，与基本hash类型将表的三个字段分别作为key、hash_key、hash_value不同，table模式下的key值可以通过“connector.table-name”和“connector.key-column”两个参数设置，将表中的所有字段名作为hash_key，字段值作为hash_value写入到hash中。

```
create table redisSink(
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_speed INT
) with (
  'connector.type' = 'redis',
  'connector.host' = 'xx.xx.xx.xx',
  'connector.port' = '6379',
  'connector.password' = 'xx',
  'connector.table-name'='car_info',
  'connector.key-column'='car_id'
);

insert into redisSink
  (car_id,car_owner,car_brand,car_speed)
  VALUES
  ("A1234","OwnA","A1234",30);
```


版权所有 © 华为技术有限公司

- 以下示例演示“connector.data-type”为string, list, hash, set类型时的建表语句。
  - “connector.data-type”为**string类型**。

    表为2列：第一列为key，第二列为value。

    ```
    create table redisSink(
      attr1 STRING,
      attr2 STRING
    ) with (
      'connector.type' = 'redis',
      'connector.host' = 'xx.xx.xx.xx',
      'connector.port' = '6379',
      'connector.password' = 'xx',
      'connector.data-type' = 'string'
    );

    insert into redisSink
      (attr1,attr2)
      VALUES
      ("car_id","A1234");
    ```

  - “connector.data-type”为**list类型**。

    表为2列：第一列为key，第二列为value。

    ```
    create table redisSink(
      attr1 STRING,
      attr2 STRING
    ) with (
      'connector.type' = 'redis',
      'connector.host' = 'xx.xx.xx.xx',
      'connector.port' = '6379',
      'connector.password' = 'xx',
      'connector.data-type' = 'list'
    );

    insert into redisSink
      (attr1,attr2)
      VALUES
      ("car_id","A1234");
    ```

  - “connector.data-type”为**set类型**。

    表为2列：第一列为key，第二列为value。

    ```
    create table redisSink(
      attr1 STRING,
      attr2 STRING
    ) with (
      'connector.type' = 'redis',
      'connector.host' = 'xx.xx.xx.xx',
      'connector.port' = '6379',
      'connector.password' = 'xx',
      'connector.data-type' = 'set'
    );

    insert into redisSink
      (attr1,attr2)
      VALUES
      ("car_id","A1234");
    ```

  - “connector.data-type”为**hash类型**。

    表为3列：第一列为key，第二列为hash_key，第三列为hash_value。

    ```
    create table redisSink(
      attr1 STRING,
      attr2 STRING,
      attr3 STRING
    ) with (
      'connector.type' = 'redis',
      'connector.host' = 'xx.xx.xx.xx',
      'connector.port' = '6379',
      'connector.password' = 'xx',
    ```

 版权所有 © 华为技术有限公司

```
  'connector.data-type' = 'hash'
);

insert into redisSink
  (attr1,attr2,attr3)
  VALUES
  ("car_info","car_id","A1234");
```

4.3.2.8 SMN 结果表

功能描述

DLI将Flink作业的输出数据输出到消息通知服务（SMN）中。

消息通知服务（Simple Message Notification，简称SMN）为DLI提供可靠的、可扩展的、海量的消息处理服务，它大大简化系统耦合，能够根据用户的需求，向订阅终端主动推送消息。可用于连接云服务、向多个协议推送消息以及集成在产生或使用通知的任何其他应用程序等场景。SMN的更多信息，请参见**《消息通知服务用户指南》**。

语法格式

```
create table smnSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'smn',
  'connector.region' = '',
  'connector.topic-urn' = '',
  'connector.message-subject' = '',
  'connector.message-column' = ''
);
```

参数说明

表 4-16 参数说明

参数	是否必选	说明
connector.type	是	sink的类型，smn表示输出到消息通知服务中
connector.region	是	SMN所在区域
connector.topic-urn	否	SMN服务的主题URN，用于静态主题URN配置。作为消息通知的目标主题，需要提前在SMN服务中创建。 与“urn_column”配置两者至少存在一个，同时配置时，“topic_urn”优先级更高。
connector.urn-column	否	主题URN内容的字段名，用于动态主题URN配置。 与“topic_urn”配置两者至少存在一个，同时配置时，“topic_urn”优先级更高。

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.message-subject	是	发送SMN服务的消息标题，用户自定义
connector.message-column	是	当前表的某个字段名，其内容作为消息的内容，用户自定义。目前只支持默认的文本消息

注意事项

无

示例

将数据写入smn的相应主题中，其中smn发送的消息的主题为'test'，内容为字段'attr1'的内容

```
create table smnSink (
  attr1 STRING,
  attr2 STRING
)
with (
  'connector.type' = 'smn',
  'connector.region' = 'cn-north-1',
  'connector.topic-urn' = 'xxxxxx',
  'connector.message-subject' = 'test',
  'connector.message-column' = 'attr1'
);
```

4.3.2.9 Hbase 结果表

功能描述

DLI将作业的输出数据输出到HBase中。HBase是一个稳定可靠，性能卓越、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。HBase支持消息数据、报表数据、推荐类数据、风控类数据、日志数据、订单数据等结构化、半结构化的KeyValue数据存储。 利用DLI，用户可方便地将海量数据高速、低时延写入HBase。

前提条件

该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。

- **若使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《 数据湖探索用户指南 》中的“**修改主机信息**”章节描述。
- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

版权所有 © 华为技术有限公司

语法格式

```
create table hbaseSink (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = '',
  'connector.zookeeper.quorum' = ''
);
```

参数说明

表 4-17 参数说明

参数	是否必选	说明
connector.type	是	connector的类型，只能为hbase
connector.version	是	该值只能为1.4.3
connector.table-name	是	hbase中的表名
connector.zookeeper.quorum	是	Zookeeper的地址
connector.zookeeper.znode.parent	否	Zookeeper中的根目录，默认是/hbase
connector.write.buffer-flush.max-size	否	每次插入的数据的最大的缓存大小，默认为2mb ,仅支持mb
connector.write.buffer-flush.max-rows	否	每次刷新数据的最大条数
connector.write.buffer-flush.interval	否	刷新时间，默认值为0s，如2s
connector.rowkey	否	设置复合rowkey，即根据多个字段设置。 形如：rowkey1:3,rowkey2:3,… 其中3表示取该字段的前3个byte，该值不能大于该字段的字节大小，且该值不能小于1

版权所有 © 华为技术有限公司

示例

```
create table hbaseSink(
 rowkey string,
 name string,
 i Row<geneder string, age int>,
 j Row<address string>
) with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = 'sink',
  'connector.rowkey' = 'rowkey:1,name:3',
  'connector.write.buffer-flush.max-rows' = '5',
  'connector.zookeeper.quorum' = 'xxxx:2181'
);
```

4.3.2.10 Elasticsearch 结果表

功能描述

DLI将Flink作业的输出数据输出到云搜索服务CSS的Elasticsearch中。Elasticsearch是基于Lucene的当前流行的企业级搜索服务器，具备分布式多用户的能力。其主要功能包括全文检索、结构化搜索、分析、聚合、高亮显示等。能为用户提供实时搜索、稳定可靠的服务。适用于日志分析、站内搜索等场景。

云搜索服务（Cloud Search Service，简称CSS）为DLI提供托管的分布式搜索引擎服务，完全兼容开源Elasticsearch搜索引擎，支持结构化、非结构化文本的多条件检索、统计、报表。云搜索服务的更多信息，请参见**《云搜索服务用户指南》**。

前提条件

- 请务必确保您的账户下已在云搜索服务里创建了集群。如何创建集群请参考《云搜索服务用户指南》中**创建集群**章节。

  如果需要通过集群账号和密码访问Elasticsearch，则创建的云搜索服务集群**必须开启安全模式并且关闭https**。
- 该场景作业需要运行在DLI的独享队列上，因此要与云搜索服务建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《数据湖探索用户指南》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《虚拟私有云用户指南》**中“安全组”章节。

注意事项

- 当前只支持CSS集群7.X及以上版本，推荐使用7.6.2版本。
- 如果不使用“connector.username”和“connector.password”参数时CSS集群请勿开启安全模式。
- CSS集群安全组入向规则必须开启ICMP。

语法格式

```
create table esSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'elasticsearch',
```

 版权所有 © 华为技术有限公司

```
 'connector.version' = '7',
 'connector.hosts' = 'http://xxxx:9200',
 'connector.index' = '',
 'connector.document-type' = '',
 'update-mode' = '',
 'format.type' = 'json'
);
```

参数说明

表 4-18 参数说明

参数	是否必选	说明
connector.type	是	connector的类型，对于elasticsearch需配置为elasticsearch
connector.version	是	使用的elasticsearch的版本。 当前只能使用版本7，即该值只能为7
connector.hosts	是	Elasticsearch所在集群的主机名，多个以’;’间隔，注意请以http开头，如http://x.x.x.x:9200
connector.index	是	Elasticsearch的索引名
connector.document-type	是	Elasticsearch的type名称 当版本为7时，由于elasticsearch使用默认的_doc类型，因此该属性无效
update-mode	是	sink的写入类型，支持append和upsert
connector.key-delimiter	否	连接复合主键的拼接符，默认为_
connector.key-null-literal	否	当key中含有null时，使用该字符代替
connector.failure-handler	否	elasticsearch请求失败时的策略，默认为fail fail：当请求失败且作业失败时抛出异常 ignore:忽略 retry-rejected:对于由于es节点的队列满时，会重新请求而不抛出失败。 custom:使用定制策略
connector.failure-handler-class	否	使用失败时的定制策略时所使用的自定义处理方式
connector.flush-on-checkpoint	否	checkpoint时是否会等待所有阻塞请求完成。 默认为true，表示会等待阻塞请求完成，如果配置为false，则表示不会等待阻塞请求完成。

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.bulk-flush.max-actions	否	批量写入时的每次最大写入记录数
connector.bulk-flush.max-size	否	批量写入时的最大数据量，当前只支持MB，请带上单位 mb
connector.bulk-flush.interval	否	批量写入时的刷新的时间间隔，单位为milliseconds，无需带上单位
format.type	是	当前只支持json
connector.username	否	Elasticsearch所在集群的账号。该账号参数需和密码“connector.password”参数同时配置。 使用账号密码参数时，创建的云搜索服务集群**必须开启安全模式并且关闭https**。
connector.password	否	Elasticsearch所在集群的密码。该密码参数需和“connector.username”参数同时配置。

示例

```
create table sink1(
  attr1 string,
  attr2 int
) with (
  'connector.type' = 'elasticsearch',
  'connector.version' = '7',
  'connector.hosts' = 'http://xxxx:9200',
  'connector.index' = 'es',
  'connector.document-type' = 'one',
  'update-mode' = 'append',
  'format.type' = 'json'
);
```

4.3.2.11 OpenTSDB 结果表

功能描述

OpenTSDB是基于HBase分布式的 ，可伸缩的时间序列数据库。OpenTSDB的设计目标是用来采集大规模集群中的监控类信息，并可实现数据的秒级查询，解决海量监控类数据在普通数据库中查询存储的局限性，可用于系统监控和测量、物联网数据、金融数据和科学实验结果数据的收集监控。

DLI可以通过增强型跨源连接功能将Flink作业的输出数据写入到OpenTSDB中。

前提条件

- 确保已经开启OpenTSDB服务。
- 该场景作业需要运行在DLI的独享队列上，因此在DLI上要与OpenTSDB建立增强型跨源连接，且用户可以根据实际所需设置相应的安全组规则。

版权所有 © 华为技术有限公司

- 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
- 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。

语法格式

```
create table tsdbSink (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = 'opentsdb',
  'connector.region' = '',
  'connector.tsdb-metrics' = '',
  'connector.tsdb-timestamps' = '',
  'connector.tsdb-values' = '',
  'connector.tsdb-tags' = '',
  'connector.tsdb-link-address' = ''
);
```

参数说明

表 4-19 参数说明

参数	是否必选	说明
connector.type	是	connector的类型，只能为opentsdb。
connector.region	是	OpenTSDB服务所在的区域。
connector.tsdb-metrics	是	数据点的metric，支持参数化。 其个数为要为1或者和“connector.tsdb-values”个数相同。 多个metric请使用“;”分隔。
connector.tsdb-timestamps	是	数据点的timestamp，仅支持指定动态列。 数据类型支持int、bigint、string，仅支持数据形式。 其个数需要为1或者和“connector.tsdb-values”的个数相同。 多个timestamp请使用“;”分隔。
connector.tsdb-values	是	数据点的value，支持指定动态列或者常数值。 多个values请使用“;”分隔。
connector.tsdb-tags	是	数据点的tags，每个tags里面至少一个标签值，最多8个标签值，多个标签使用“,”分隔，支持参数化。 其个数需要为1或者和“connector.tsdb-values”的个数相同。 多个tags请使用“;”分隔。
connector.batch-insert-data-num	否	表示一次性批量写入的数据量，即数据条数，值必须为正整数，默认值为8。

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.tsdb-link-address	是	待插入数据所属集群的OpenTSDB连接地址。

注意事项

- 若使用MRS集群的OpenTSDB，请确保以下几点：
  a. OpenTSDB的ip地址和端口请从OpenTSDB服务配置中查看配置项“tsd.network.bind”和“tsd.network.port”分别获取。
  b. 若OpenTSDB服务配置项“tsd.https.enabled”的值为true，则sql语句中的“connector.tsdb-link-address”参数值格式为https://ip:port。若“tsd.https.enabled”为false，则“connector.tsdb-link-address”参数值格式可以为http://ip:port或者ip:port。
  c. 在建立增强型跨源连接时，需要将MRS集群中的/etc/hosts主机和ip映射信息添加到“主机信息”参数中。
- 当配置项支持参数化时，表示将记录中的一列或者多列作为该配置项的一部分。例如当配置项设置为car_${car_brand}时，如果一条记录的car_brand列值为BMW，则该配置项在该条记录下为car_BMW。
- 若支持动态列，则其形式需要为${*columnName*}，其中columnName为相应的字段名。

示例

```
create table sink1(
  attr1 bigint,
  attr2 int,
  attr3 int
) with (
  'connector.type' = 'opentsdb',
  'connector.region' = '',
  'connector.tsdb-metrics' = '',
  'connector.tsdb-timestamps' = '${attr1}',
  'connector.tsdb-values' = '${attr2};10',
  'connector.tsdb-tags' = 'key1:value1,key2:value2;key3:value3',
  'connector.tsdb-link-address' = ''
);
```

4.3.2.12 userDefined 结果表

功能描述

您可通过编写代码实现将DLI处理之后的数据写入到指定的云生态或者开源生态。

前提条件

已编写代码实现自定义sink类：

自定义sink类需要继承Flink开源类：RichSinkFunction，并指定数据类型为：Tuple2<Boolean, Row>。

 版权所有 © 华为技术有限公司

例如开发自定义类MySink：**public class MySink extends RichSinkFunction<** Tuple2<Boolean, Row>**>{}**，需重点实现其中的open、invoke和close函数。代码参考示例如下：

```
public class MySink extends RichSinkFunction<Tuple2<Boolean, Row>> {
    // 初始化
    @Override
    public void open(Configuration parameters) throws Exception {}

    @Override
    //业务数据处理逻辑具体实现
    /*in包括两个值，其中第一个值为布尔型，为true或false，当true时表示插入或更新操作，为false时表示删除操作，若对接的sink端不支持删除等操作，当为false时，可不进行任何操作。第二个值表示实际的数据值*/
    public void invoke(Tuple2<Boolean, Row> in, Context context) throws Exception {}

    @Override
    public void close() throws Exception {}
}
```

依赖的pom配置文件内容参考如下：

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java_2.11</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-core</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
```

实现完成后将该类编译打包在Jar中，通过Flink OpenSource SQL作业编辑页的UDF Jar参数上传。具体Flink OpenSource SQL作业创建指导可以参考**https://support.huaweicloud.com/usermanual-dli/dli_01_0498.html**。

语法格式

```
create table userDefinedSink (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = ''
);
```

参数说明

表 4-20 参数说明

参数	是否必选	说明
connector.type	是	只能为user-defined，表示使用自定义的sink。
connector.class-name	是	sink函数的全限定类名。sink类的具体实现可以参考**前提条件**说明。
connector.class-parameter	否	sink函数其构造函数的参数，只支持一个String类型的参数。

 版权所有 © 华为技术有限公司

注意事项

connector.class-name需要为全限定类名。

示例

```
create table userDefinedSink (
  attr1 int,
  attr2 int
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = 'xx.xx.MySink'
);
```

4.3.2.13 Print 结果表

功能描述

print connector用于将用户输出的数据打印到error文件或者taskmanager的out文件中，方便用户查看，主要用于代码调试，查看输出结果。

语法格式

```
create table printSink (
  attr_name attr_type (',' attr_name attr_type) * (',' PRIMARY KEY (attr_name,...) NOT ENFORCED)
) with (
  'connector' = 'print',
  'print-identifier' = '',
  'standard-error' = ''
);
```

参数说明

表 4-21 参数说明

参数	是否必选	说明
connector	是	固定为print。
print-identifier	否	配置一个标识符作为输出数据的前缀。
standard-error	否	该值只能为true或false，默认为false。 ● 若为true，则表示输出数据到taskmanager的error文件中。 ● 若为false，则表示输出数据到taskmanager的out中。

 版权所有 © 华为技术有限公司

示例

从kafka中读取数据输出到taskmanager的out文件中，可以在taskmanager的out文件中看到输出结果。

```
create table kafkaSource(
  attr0 string,
  attr1 boolean,
  attr3 decimal(38, 18),
  attr4 TINYINT,
  attr5 smallint,
  attr6 int,
  attr7 bigint,
  attr8 float,
  attr9 double,
  attr10 date,
  attr11 time,
  attr12 timestamp(3)
) with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test_json',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.properties.group.id' = 'test_print',
  'connector.startup-mode' = 'latest-offset',
  'format.type' = 'csv'
);

create table printTable(
  attr0 string,
  attr1 boolean,
  attr3 decimal(38,18),
  attr4 TINYINT,
  attr5 smallint,
  attr6 int,
  attr7 bigint,
  attr8 float,
  attr9 double,
  attr10 date,
  attr11 time,
  attr12 timestamp(3),
  attr13 array<string>,
  attr14 row<attr15 float, attr16 timestamp(3)>,
  attr17 map<int, bigint>
) with (
  "connector" = "print"
);

insert into
  printTable
select
  attr0,
  attr1,
  attr3,
  attr4,
  attr5,
  attr6,
  attr7,
  attr8,
  attr9,
  attr10,
  attr11,
  attr12,
  array [cast(attr0 as string), cast(attr0 as string)],
  row(
    cast(attr8 as float),
    cast(attr12 as timestamp(3))
  ),
  map [cast(attr6 as int), cast(attr7 as bigint)]
```

 版权所有 © 华为技术有限公司

```
from
  kafkaSource;
```

4.3.2.14 FileSytem 结果表

功能描述

FileSystem结果表用于将数据输出到分布式文件系统HDFS或者对象存储服务OBS等文件系统。数据生成后，可直接对生成的目录创建非DLI表，通过DLI SQL进行下一步处理分析，并且输出数据目录支持分区表结构。适用于数据转储、大数据分析、备份或活跃归档、深度或冷归档等场景。

语法格式

```
create table filesystemSink (
  attr_name attr_type (',' attr_name attr_type) *
) with (
  'connector.type' = 'filesystem',
  'connector.file-path' = '',
  'format.type' = ''
);
```

注意事项

- 该建表语法的数据输出目录为OBS时，OBS必须为并行文件系统，不能为OBS桶。
- 使用fileSystem时必须开启checkpoint，保证作业的一致性。
- format.type为parquet时，支持的数据类型为string, boolean, tinyint, smallint, int, bigint, float, double, map<string, string>, timestamp(3), time。
- 为了避免数据丢失或者数据被覆盖，开启作业异常自动重启，需要配置为“从checkpoint恢复”。
- checkpoint间隔设置需在输出文件实时性、文件大小和恢复时长之间进行权衡，比如10分钟。
- 使用HDFS时需要绑定相应的跨源，并填写相应的主机信息。
- 使用hdfs时，请配置主NameNode的所在节点信息。

参数说明

表 4-22 参数说明

参数	是否必选	说明
connector.type	是	固定为filesystem。

 版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.file-path	是	数据输出目录，格式为: *schema*://*file.path*。 **说明** 当前schame只支持obs和hdfs。 • 当schema为obs时，表示输出到对象存储服务OBS。**注意，OBS必须是并行文件系统，不能是OBS桶。** 示例：obs://bucketName/fileName，表示数据输出到obs的bucketName桶下的fileName目录中。 • 当schema为hdfs时，表示输出到HDFS。 示例：hdfs://node-master1sYAx:9820/user/car_infos，其中node-master1sYAx:9820为MRS集群NameNode所在节点信息。
format.type	是	输出数据编码格式，当前支持“parquet”格式和“csv”格式。 • 当schema为obs时，输出数据编码格式仅支持“parquet”格式。 • 当schema为hdfs时，输出数据编码格式支持“parquet”格式和“csv”格式。
format.field-delimiter	否	属性分隔符。 当编码格式为“csv”时，需要设置属性分隔符，用户可以自定义，默认为“,”。
connector.ak	否	用于访问obs的accessKey 当写入obs时必须填写该字段。
connector.sk	否	用于访问obs的secretKey 当写入obs时必须填写该字段。
connector.partitioned-by	否	分区字段，多个字段以“,”分隔

示例

从kafka中读取数据以parquet的格式写到obs的bucketName桶下的fileName目录中。

```
create table kafkaSource(
  attr0 string,
  attr1 boolean,
  attr2 TINYINT,
  attr3 smallint,
  attr4 int,
  attr5 bigint,
  attr6 float,
  attr7 double,
  attr8 timestamp(3),
  attr9 time
) with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'test_json',
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
```


版权所有 © 华为技术有限公司

```
  'connector.properties.group.id' = 'test_filesystem',
  'connector.startup-mode' = 'latest-offset',
  'format.type' = 'csv'
);

create table filesystemSink(
  attr0 string,
  attr1 boolean,
  attr2 TINYINT,
  attr3 smallint,
  attr4 int,
  attr5 bigint,
  attr6 float,
  attr7 double,
  attr8 map < string,  string >,
  attr9 timestamp(3),
  attr10 time
) with (
  "connector.type" = "filesystem",
  "connector.file-path" = "obs://bucketName/fileName",
  "format.type" = "parquet",
  "connector.ak" = "xxxx",
  "connector.sk" = "xxxxxx"
);

insert into
  filesystemSink
select
  attr0,
  attr1,
  attr2,
  attr3,
  attr4,
  attr5,
  attr6,
  attr7,
  map [attr0,attr0],
  attr8,
  attr9
from
  kafkaSource;
```

4.3.3 创建维表

4.3.3.1 创建 JDBC 维表

创建JDBC表用于与输入流连接。

前提条件

- 请务必确保您的账户下已创建了相应实例。

语法格式

```
CREATE TABLE  table_id (
  attr_name attr_type
  (',' attr_name attr_type)*
)
  WITH (
  'connector.type' = 'jdbc',
  'connector.url' = '',
  'connector.table' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

 版权所有 © 华为技术有限公司

参数说明

表 4-23 参数说明

参数	是否必选	说明
connector.type	是	数据源类型，‘jdbc’表示使用JDBC connector，必须为jdbc
connector.url	是	数据库的URL
connector.table	是	读取数据库中的数据所在的表名
connector.driver	否	连接数据库所需要的驱动。若未配置，则会自动通过URL提取
connector.username	否	数据库认证用户名，需要和'connector.password'一起配置
connector.password	否	数据库认证密码，需要和'connector.username'一起配置
connector.read.partition.column	否	用于对输入进行分区的列名 与connector.read.partition.lower-bound、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.lower-bound	否	第一个分区的最小值 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.upper-bound	否	最后一个分区的最大值 与connector.read.partition.column、connector.read.partition.lower-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.num	否	分区的个数 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.upper-bound必须同时存在或者同时不存在
connector.read.fetch-size	否	每次从数据库拉取数据的行数。默认值为0，表示忽略该提示。

版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.lookup.cache.max-rows	否	维表配置，缓存的最大行数，超过该值时，最先添加的数据将被标记为过期。-1表示不使用缓存。
connector.lookup.cache.ttl	否	维表配置，缓存超时时间，超过该时间的数据会被剔除。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
connector.lookup.max-retries	否	维表配置，数据拉取最大重试次数，默认为3。

示例

RDS表用于与输入流连接。

```
CREATE TABLE car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  proctime as PROCTIME()
)
  WITH (
  'connector.type' = 'dis',
  'connector.region' = 'cn-north-1',
  'connector.channel' = 'disInput',
  'format.type' = 'csv'
  );

CREATE TABLE  db_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
  'connector.type' = 'jdbc',
  'connector.url' = 'jdbc:mysql://xx.xx.xx.xx:3306/xx',
  'connector.table' = 'jdbc_table_name',
  'connector.driver' = 'com.mysql.jdbc.Driver',
  'connector.username' = 'xxx',
  'connector.password' = 'xxxxx'
);

CREATE TABLE audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
  'connector.type' = 'dis',
  'connector.region' = 'cn-north-1',
  'connector.channel' = 'disOutput',
  'connector.partition-key' = 'car_id,car_owner',
  'format.type' = 'csv'
  );
```


版权所有 © 华为技术有限公司

```
INSERT INTO audi_cheaper_than_30w
SELECT a.car_id, b.car_owner, b.car_brand, b.car_price
FROM car_infos as a join db_info FOR SYSTEM_TIME AS OF a.proctime AS b on a.car_id = b.car_id;
```

4.3.3.2 创建 DWS 维表

创建DWS表用于与输入流连接。

前提条件

- 请务必确保您的账户下已创建了所需的DWS实例。

语法格式

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector.type' = 'gaussdb',
  'connector.url' = '',
  'connector.table' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

参数说明

表 4-24 参数说明

参数	是否必选	说明
connector.type	是	connector类型，需配置为'gaussdb'
connector.url	是	jdbc连接地址，格式为：jdbc:postgresql://${ip}:${port}/${dbName} 。
connector.table	是	读取数据库中的数据所在的表名
connector.driver	否	jdbc连接驱动，默认为: org.postgresql.Driver。
connector.username	否	数据库认证用户名，需要和'connector.password'一起配置
connector.password	否	数据库认证密码，需要和'connector.username'一起配置
connector.read.partition.column	否	用于对输入进行分区的列名 与connector.read.partition.lower-bound、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在

 版权所有 © 华为技术有限公司

参数	是否必选	说明
connector.read.partition.lower-bound	否	第一个分区的最小值 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.upper-bound	否	最后一个分区的最大值 与connector.read.partition.column、connector.read.partition.lower-bound、 connector.read.partition.num必须同时存在或者同时不存在
connector.read.partition.num	否	分区的个数 与connector.read.partition.column、connector.read.partition.upper-bound、 connector.read.partition.upper-bound必须同时存在或者同时不存在
connector.read.fetch-size	否	每次从数据库拉取数据的行数。默认值为0，表示忽略该提示
connector.lookup.cache.max-rows	否	维表配置，缓存的最大行数，超过该值时，最先添加的数据将被标记为过期。-1表示不使用缓存。
connector.lookup.cache.ttl	否	维表配置，缓存超时时间，超过该时间的数据会被剔除。格式为：{length value}{time unit label}，如123ms, 321s，支持的时间单位包括: d,h,min,s,ms等，默认为ms。
connector.lookup.max-retries	否	维表配置，数据拉取最大重试次数，默认为3。

示例

RDS表用于与输入流连接。

```
CREATE TABLE car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  proctime as PROCTIME()
)
  WITH (
  'connector.type' = 'dis',
  'connector.region' = 'cn-north-1',
  'connector.channel' = 'disInput',
  'format.type' = 'csv'
  );

CREATE TABLE  db_info (
```


版权所有 © 华为技术有限公司

```
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
  'connector.type' = 'gaussdb',
  'connector.driver' = 'org.postgresql.Driver',
  'connector.url' = 'jdbc:gaussdb://xx.xx.xx.xx:8000/xx',
  'connector.table' = 'car_info',
  'connector.username' = 'xx',
  'connector.password' = 'xx',
  'connector.lookup.cache.max-rows' = '10000',
  'connector.lookup.cache.ttl' = '24h'
);

CREATE TABLE audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
  'connector.type' = 'dis',
  'connector.region' = 'cn-north-1',
  'connector.channel' = 'disOutput',
  'connector.partition-key' = 'car_id,car_owner',
  'format.type' = 'csv'
  );

INSERT INTO audi_cheaper_than_30w
SELECT a.car_id, b.car_owner, b.car_brand, b.car_price
FROM car_infos as a join db_info FOR SYSTEM_TIME AS OF a.proctime AS b on a.car_id = b.car_id;
```

4.3.3.3 创建 Hbase 维表

功能描述

创建Hbase维表用于与输入流连接。

前提条件

- 该场景作业需要运行在DLI的独享队列上，因此要与HBase建立增强型跨源连接，且用户可以根据实际所需设置相应安全组规则。
  - 如何建立增强型跨源连接，请参考《 数据湖探索用户指南 》中**增强型跨源连接**章节。
  - 如何设置安全组规则，请参见**《 虚拟私有云用户指南 》**中“安全组”章节。
- **若使用MRS HBase，请在增强型跨源的主机信息中添加MRS集群所有节点的主机ip信息。**

  详细操作请参考《 数据湖探索用户指南 》中的“**修改主机信息**”章节描述。

语法格式

```
create table hbaseSource (
  attr_name attr_type
  (',' attr_name attr_type)*
 )
with (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',
  'connector.table-name' = '',
```

 版权所有 © 华为技术有限公司

```
  'connector.zookeeper.quorum' = ''
);
```

参数说明

表 4-25 参数说明

参数	是否必选	说明
connector.type	是	connector的类型，只能为hbase
connector.version	是	该值只能为1.4.3
connector. table-name	是	hbase中的表名
connector.zookeeper.quorum	是	Zookeeper的地址
connector.zookeeper.znode.parent	否	Zookeeper中的根目录，默认是/hbase

示例

```
create table hbaseSource(
  id string,
  i Row<score string>
 ) with (
   'connector.type' = 'hbase',
   'connector.version' = '1.4.3',
   'connector.table-name' = 'user',
   'connector.zookeeper.quorum' = 'xxxx:2181'
 );
create table source1(
  id string,
  name string,
  geneder string,
  age int,
  address string,
  proctime as PROCTIME()
) with (
  "connector.type" = "dis",
  "connector.region" = "cn-north-1",
  "connector.channel" = "read",
  "connector.ak" = "xxxxxx",
  "connector.sk" = "xxxxxx",
  "format.type" = 'csv'
);

 create table hbaseSink(
  rowkey string,
  i Row<name string, geneder string, age int, address string>,
  j ROW<score string>
 ) with (
   'connector.type' = 'hbase',
   'connector.version' = '1.4.3',
   'connector.table-name' = 'score',
   'connector.write.buffer-flush.max-rows' = '1',
   'connector.zookeeper.quorum' = 'xxxx:2181'
 );
```


版权所有 © 华为技术有限公司

```
 insert into hbaseSink select d.id, ROW(name, geneder,age,address), ROW(score) from source1 as d join
hbaseSource for system_time as of d.proctime as h on d.id = h.id;
```

4.4 数据操作语句 DML

4.4.1 SELECT

SELECT

语法格式

```
SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

SELECT语句用于从表中选取数据。

ALL表示返回所有结果。

DISTINCT表示返回不重复结果。

注意事项

- 所查询的表必须是已经存在的表，否则会出错。
- WHERE关键字指定查询的过滤条件，过滤条件中支持算术运算符，关系运算符，逻辑运算符。
- GROUP BY指定分组的字段，可以单字段分组，也可以多字段分组。

示例

找出数量超过3的订单。

```
insert into temp SELECT  * FROM Orders WHERE units > 3;
```

插入一组常量数据。

```
insert into temp select 'Lily’, ‘male’, ‘student’, 17;
```

WHERE 过滤子句

语法格式

```
SELECT  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
```

语法说明

利用WHERE子句过滤查询结果。

注意事项

- 所查询的表必须是已经存在的，否则会出错。
- WHERE条件过滤，将不满足条件的记录过滤掉，返回满足要求的记录。

 版权所有 © 华为技术有限公司

示例

找出数量超过3并且小于10的订单。

```
insert into temp SELECT  * FROM Orders
  WHERE units > 3 and units < 10;
```

HAVING 过滤子句

功能描述

利用HAVING子句过滤查询结果。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤，HAVING子句支持算术运算，聚合函数等。

注意事项

如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。

示例

根据字段name对表student进行分组，再按组将score最大值大于95的记录筛选出来。

```
insert into temp SELECT name, max(score) FROM student
  GROUP BY name
  HAVING max(score) >95;
```

按列 GROUP BY

功能描述

按列进行分组操作。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
```

语法说明

GROUP BY：按列可分为单列GROUP BY与多列GROUP BY。

- 单列GROUP BY：指GROUP BY子句中仅包含一列。
- 多列GROUP BY：指GROUP BY子句中不止一列，查询语句将按照GROUP BY的所有字段分组，所有字段都相同的记录将被放在同一组中。

注意事项

GroupBy在流处理表中会产生更新结果

 版权所有 © 华为技术有限公司

示例

根据score及name两个字段对表student进行分组，并返回分组结果。

```
insert into temp SELECT name,score, max(score) FROM student
  GROUP BY name,score;
```

表达式 GROUP BY

功能描述

按表达式对流进行分组操作。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
```

语法说明

groupItem：可以是单字段，多字段，也可以是字符串函数等调用，不能是聚合函数。

注意事项

无

示例

先利用substring函数取字段name的子字符串，并按照该子字符串进行分组，返回每个子字符串及对应的记录数。

```
insert into temp SELECT substring(name,6),count(name) FROM student
  GROUP BY substring(name,6);
```

Grouping sets, Rollup, Cube

功能描述

- GROUPING SETS 的 GROUP BY 子句可以生成一个等效于由多个简单 GROUP BY 子句的 UNION ALL 生成的结果集，并且其效率比 GROUP BY 要高。
- ROLLUP与CUBE按一定的规则产生多种分组，然后按各种分组统计数据。
- CUBE生成的结果集显示了所选列中值的所有组合的聚合。
- Rollup生成的结果集显示了所选列中值的某一层次结构的聚合。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY groupingItem]
```

语法说明

groupingItem：是Grouping sets(columnName [, columnName]*)、Rollup(columnName [, columnName]*)、Cube(columnName [, columnName]*)

注意事项

无

示例

 版权所有 © 华为技术有限公司

分别产生基于user和product的结果

```
INSERT INTO temp SELECT SUM(amount)
FROM Orders
GROUP BY GROUPING SETS ((user), (product));
```

GROUP BY 中使用 HAVING 过滤

功能描述

利用HAVING子句在表分组后实现过滤。

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
```

语法说明

HAVING：一般与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。

注意事项

- 如果过滤条件受GROUP BY的查询结果影响，则不能用WHERE子句进行过滤，而要用HAVING子句进行过滤。HAVING与GROUP BY合用，先通过GROUP BY进行分组，再在HAVING子句中进行过滤。
- HAVING中除聚合函数外所使用的字段必须是GROUP BY中出现的字段。
- HAVING子句支持算术运算，聚合函数等。

示例

先依据num对表transactions进行分组，再利用HAVING子句对查询结果进行过滤，price与amount乘积的最大值大于5000的记录将被筛选出来，返回对应的num及price与amount乘积的最大值。

```
insert into temp SELECT num, max(price*amount) FROM transactions
  WHERE time > '2016-06-01'
  GROUP BY num
  HAVING max(price*amount)>5000;
```

4.4.2 集合操作

Union/Union ALL/Intersect/Except

语法格式

```
query UNION [ ALL ] | Intersect | Except query
```

语法说明

- UNION返回多个查询结果的并集。
- Intersect返回多个查询结果的交集。
- Except返回多个查询结果的差集。

注意事项

 版权所有 © 华为技术有限公司

- 集合运算是以一定条件将表首尾相接，所以其中每一个SELECT语句返回的列数必须相同，列的类型一定要相同，列名不一定要相同。
- UNION默认是去重的，UNION ALL是不去重的。

示例

输出Orders1和Orders2的并集，不包含重复记录。

```
insert into temp SELECT  * FROM Orders1
  UNION SELECT  * FROM Orders2;
```

IN

语法格式

```
SELECT [ ALL | DISTINCT ]  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  WHERE column_name IN (value (, value)* ) | query
```

语法说明

IN操作符允许在where子句中规定多个值。若表达式在给定的表子查询中存在，则返回 true 。

注意事项

子查询表必须由单个列构成，且该列的数据类型需与表达式保持一致。

示例

输出Orders中NewProducts中product的user和amount信息。

```
insert into temp SELECT user, amount
FROM Orders
WHERE product IN (
    SELECT product FROM NewProducts
);
```

4.4.3 窗口

GROUP WINDOW

语法说明

Group Window定义在GROUP BY里，每个分组只输出一条记录，包括以下几种：

- 分组函数

> **⚠ 注意**
>
> - 在流处理表中的 SQL 查询中，分组窗口函数的 time_attr 参数必须引用一个合法的时间属性，且该属性需要指定行的处理时间或事件时间。
> - 对于批处理的 SQL 查询，分组窗口函数的 time_attr 参数必须是一个 TIMESTAMP 类型的属性。

版权所有 © 华为技术有限公司

表 4-26 分组函数表

分组窗口函数	说明
TUMBLE(time_attr, interval)	定义一个滚动窗口。滚动窗口把行分配到有固定持续时间（ interval ）的不重叠的连续窗口。比如，5 分钟的滚动窗口以 5 分钟为间隔对行进行分组。滚动窗口可以定义在事件时间（批处理、流处理）或处理时间（流处理）上。
HOP(time_attr, interval, interval)	定义一个跳跃的时间窗口（在 Table API 中称为滑动窗口）。滑动窗口有一个固定的持续时间（ 第二个 interval 参数 ）以及一个滑动的间隔（第一个 interval 参数 ）。若滑动间隔小于窗口的持续时间，滑动窗口则会出现重叠；因此，行将会被分配到多个窗口中。比如，一个大小为 15 分组的滑动窗口，其滑动间隔为 5 分钟，将会把每一行数据分配到 3 个 15 分钟的窗口中。滑动窗口可以定义在事件时间（批处理、流处理）或处理时间（流处理）上。
SESSION(time_attr, interval)	定义一个会话时间窗口。会话时间窗口没有一个固定的持续时间，但是它们的边界会根据 interval 所定义的不活跃时间所确定；即一个会话时间窗口在定义的间隔时间内没有时间出现，该窗口会被关闭。例如时间窗口的间隔时间是 30 分钟，当其不活跃的时间达到30分钟后，若观测到新的记录，则会启动一个新的会话时间窗口（否则该行数据会被添加到当前的窗口），且若在 30 分钟内没有观测到新记录，这个窗口将会被关闭。会话时间窗口可以使用事件时间（批处理、流处理）或处理时间（流处理）。

- 窗口辅助函数

  可以使用以下辅助函数选择组窗口的开始和结束时间戳以及时间属性。

⚠ 注意

辅助函数必须使用与GROUP BY 子句中的分组窗口函数完全相同的参数来调用

表 4-27 窗口辅助函数表

辅助函数	说明
TUMBLE_START(time_attr, interval) HOP_START(time_attr, interval, interval) SESSION_START(time_attr, interval)	返回相对应的滚动、滑动和会话窗口范围内的下界时间戳。

版权所有 © 华为技术有限公司

辅助函数	说明
TUMBLE_END(time_attr, interval) HOP_END(time_attr, interval, interval) SESSION_END(time_attr, interval)	返回相对应的滚动、滑动和会话窗口*范围以外*的上界时间戳。 注意： 范围以外的上界时间戳*不可以*在随后基于时间的操作中，作为行时间属性使用，比如基于时间窗口的join以及分组窗口或分组窗口上的聚合。
TUMBLE_ROWTIME(time_attr, interval) HOP_ROWTIME(time_attr, interval, interval) SESSION_ROWTIME(time_attr, interval)	返回的是一个可用于后续需要基于时间的操作的时间属性（rowtime attribute），比如基于时间窗口的join以及 分组窗口或分组窗口上的聚合。
TUMBLE_PROCTIME(time_attr, interval) HOP_PROCTIME(time_attr, interval, interval) SESSION_PROCTIME(time_attr, interval)	返回一个可用于后续需要基于时间的操作的 处理时间参数，比如基于时间窗口的join以及分组窗口或分组窗口上的聚合.

示例

```
// 每天计算SUM（金额）（事件时间）。
insert into temp SELECT name,
    TUMBLE_START(ts, INTERVAL '1' DAY) as wStart,
    SUM(amount)
    FROM Orders
    GROUP BY TUMBLE(ts, INTERVAL '1' DAY), name;

// 每天计算SUM（金额）（处理时间）。
insert into temp SELECT name,
    SUM(amount)
    FROM Orders
    GROUP BY TUMBLE(proctime, INTERVAL '1' DAY), name;

// 每个小时计算事件时间中最近24小时的SUM（数量）。
insert into temp SELECT product,
    SUM(amount)
    FROM Orders
    GROUP BY HOP(ts, INTERVAL '1' HOUR, INTERVAL '1' DAY), product;

// 计算每个会话的SUM（数量），间隔12小时的不活动间隙（事件时间）。
insert into temp SELECT name,
    SESSION_START(ts, INTERVAL '12' HOUR) AS sStart,
    SESSION_END(ts, INTERVAL '12' HOUR) AS sEnd,
    SUM(amount)
    FROM Orders
    GROUP BY SESSION(ts, INTERVAL '12' HOUR), name;
```

TUMBLE WINDOW 扩展

功能描述

DLI TUMBLE函数功能增强主要包括以下功能：

版权所有 © 华为技术有限公司

- TUMBLE窗口周期性触发，控制延迟

  TUMBLE窗口结束之前，可以根据设置的触发频率周期性地触发窗口，输出从窗口开始时间到当前周期时间窗口内的计算结果值，但不影响最终窗口输出值，从而在窗口结束前的每个周期都可以看到最新的结果。

- 提高数据的精确性

  在窗口结束后，允许设置延迟时间。根据设置的延迟时间，每到达一个迟到数据，则更新窗口的输出结果

注意事项

若使用insert语句将结果写入sink中，则sink需要支持upsert模式。

语法格式

```
TUMBLE(time_attr, window_interval, period_interval, lateness_interval)
```

语法示例

例如当前time_attr属性列为：testtime，窗口时间间隔为10秒，语法示例为：

```
TUMBLE(testtime, INTERVAL '10' SECOND, INTERVAL '10' SECOND, INTERVAL '10' SECOND)
```

参数说明

表 4-28 参数说明

参数	说明	参数格式
time_attr	表示相应的事件时间或者处理时间属性列。	-
window_interval	表示窗口的持续时长。	• 格式1：**INTERVAL** '10' **SECOND** 表示窗口时间间隔为10秒，请根据实际情况修改该时间值。 • 格式2：**INTERVAL** '10' **MINUTE** 表示窗口时间间隔为10分钟，请根据实际情况修改该时间值。 • 格式3：**INTERVAL** '10' **DAY** 表示窗口时间间隔为10天，请根据实际情况修改该时间值。
period_interval	表示在窗口范围内周期性触发的频率，即在窗口结束前，从窗口开启开始，每隔period_interval时长更新一次输出结果。若没有设置，则默认没有使用周期触发策略。	
lateness_interval	表示窗口结束后延迟lateness_interval时长，继续统计在窗口结束后延迟时间内到达的属于该窗口的数据，而且在延迟时间内到达的每个数据都会更新输出结果。 **说明** 当时间窗口为处理时间时，无论lateness_interval为何值，都不会有效果。	

 版权所有 © 华为技术有限公司

说明

period_interval和lateness_interval不可为负数。

- 当period_interval为0时，表示没有使用窗口的周期触发策略；
- 当lateness_interval为0时，表示没有使用窗口结束后的延迟策略；
- 当二者都没有填写时，默认两种策略都没有配置，仅使用普通的TUMBLE窗口。
- 若仅需使用延迟时间策略，则需要将上述period_interval格式中的'10'设置为 '0'。

OVER WINDOW

Over Window与Group Window区别在于Over window每一行都会输出一条记录。

语法格式

```
SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  ROWS
 BETWEEN (UNBOUNDED|rowCOUNT) PRECEDING AND CURRENT ROW FROM TABLENAME

SELECT agg1(attr1) OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime
  RANGE
  BETWEEN (UNBOUNDED|timeInterval) PRECEDING AND CURRENT ROW FROM TABLENAME
```

语法说明

表 4-29 参数说明

参数	参数说明
PARTITION BY	指定分组的主键，每个分组各自进行计算。
ORDER BY	指定数据按processing time或event time作为时间戳。
ROWS	个数窗口。
RANGE	时间窗口。

注意事项

- 所有的聚合必须定义到同一个窗口中，即相同的分区、排序和区间。
- 当前仅支持 PRECEDING (无界或有界) 到 CURRENT ROW 范围内的窗口、FOLLOWING 所描述的区间并未支持。
- ORDER BY 必须指定于单个的时间属性。

示例

```
// 计算从规则启动到目前为止的计数及总和(in proctime)
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as
cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY proctime RANGE UNBOUNDED preceding) as cnt2
    FROM Orders;
```

 版权所有 © 华为技术有限公司

```
// 计算最近四条记录的计数及总和(in proctime)
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY proctime ROWS BETWEEN 4 PRECEDING AND
CURRENT ROW) as cnt2
    FROM Orders;

// 计算最近60s的计数及总和(in eventtime),基于事件时间处理，事件时间为Orders中的timeattr字段。
insert into temp SELECT name,
    count(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60'
SECOND PRECEDING AND CURRENT ROW) as cnt1,
    sum(amount) OVER (PARTITION BY name ORDER BY timeattr RANGE BETWEEN INTERVAL '60' SECOND
PRECEDING AND CURRENT ROW) as cnt2
    FROM Orders;
```

4.4.4 JOIN

Equi-join

语法格式

```
FROM tableExpression INNER | LEFT | RIGHT | FULL JOIN tableExpression
  ON value11 = value21 [ AND value12 = value22]
```

注意事项

- 目前仅支持 equi-join ，即 join 的联合条件至少拥有一个相等谓词。不支持任何 cross join 和 theta join。
- Join 的顺序没有进行优化，join 会按照 FROM 中所定义的顺序依次执行。请确保 join 所指定的表在顺序执行中不会产生不支持的 cross join （笛卡儿积）以致查询失败。
- 流查询中可能会因为不同行的输入数量导致计算结果的状态无限增长。请提供具有有效保留间隔的查询配置，以防止出现过多的状态。

示例

```
SELECT *
FROM Orders INNER JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders LEFT JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders RIGHT JOIN Product ON Orders.productId = Product.id;

SELECT *
FROM Orders FULL OUTER JOIN Product ON Orders.productId = Product.id;
```

Time-windowed Join

功能描述

每条流的每一条数据会与另一条流上的不同时间区域的数据进行JOIN。

语法格式

```
from t1 JOIN t2 ON t1.key = t2.key AND TIMEBOUND_EXPRESSIO
```

语法描述

TIMEBOUND_EXPRESSION 有两种写法，如下：

 版权所有 © 华为技术有限公司

- L.time between LowerBound(R.time) and UpperBound(R.time)
- R.time between LowerBound(L.time) and UpperBound(L.time)
- 带有时间属性(L.time/R.time)的比较表达式。

注意事项

时间窗口join需要至少一个 equi-join 谓词和一个限制了双方时间的 join 条件。

例如使用两个适当的范围谓词（<, <=, >=, >），一个 BETWEEN 谓词或一个比较两个输入表中相同类型的时间属性（即处理时间和事件时间）的相等谓词

比如，以下谓词是合法的窗口 join 条件：

- ltime = rtime
- ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
- ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND

示例

所有在收到后四小时内发货的 order 会与它们相关的 shipment 进行 join。

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.orderId AND
    o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime;
```

Expanding arrays into a relation

注意事项

目前尚未支持非嵌套的 WITH ORDINALITY 。

示例

```
SELECT users, tag
FROM Orders CROSS JOIN UNNEST(tags) AS t (tag);
```

Join 表函数(UDTF)

功能描述

将表与表函数的结果进行 join 操作。左表（outer）中的每一行将会与调用表函数所产生的所有结果中相关联行进行 join 。

注意事项

针对横向表的左外部连接当前仅支持文本常量 TRUE 作为谓词。

示例

若表函数返回了空结果，左表（outer）的行将会被删除

```
SELECT users, tag
FROM Orders, LATERAL TABLE(unnest_udtf(tags)) t AS tag;
```

若表函数返回了空结果，将会保留相对应的外部行并用空值填充

```
SELECT users, tag
FROM Orders LEFT JOIN LATERAL TABLE(unnest_udtf(tags)) t AS tag ON TRUE;
```

 版权所有 © 华为技术有限公司

Join Temporal Table Function

功能描述

注意事项

目前仅支持在 Temporal Tables 上的 inner join

示例

假如Rates是一个 Temporal Table Function， join 可以使用 SQL 进行如下的表达:

```
SELECT
  o_amount, r_rate
FROM
  Orders,
  LATERAL TABLE (Rates(o_proctime))
WHERE
  r_currency = o_currency;
```

Join Temporal Tables

功能描述

与Temporal表进行join操作

语法格式

```
SELECT column-names
FROM table1   [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF table1.proctime [AS <alias2>]
ON table1.column-name1 = table2.key-name1
```

语法说明

- table1.proctime表示table1的proctime处理时间属性(计算列)
- 使用FOR SYSTEM_TIME AS OF table1.proctime表示当左边表的记录与右边的维表join时，只匹配当前处理时间维表所对应的的快照数据。

注意事项

仅支持带有处理时间的 temporal tables 的 inner 和 left join

示例

假设 LatestRates 是一个根据最新的 rates 物化的Temporal Table。

```
SELECT
  o.amout, o.currency, r.rate, o.amount * r.rate
FROM
  Orders AS o
  JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
  ON r.currency = o.currency;
```

4.4.5 OrderBy & Limit

OrderBy

功能描述

主要根据时间属性按照升序进行排序

注意事项

 版权所有 © 华为技术有限公司

目前仅支持根据时间属性进行排序

示例

对订单根据订单时间进行升序排序

```
SELECT *
FROM Orders
ORDER BY orderTime;
```

Limit

功能描述

限制返回的数据结果个数

注意事项

LIMIT 查询需要有一个 ORDER BY 字句

示例

```
SELECT *
FROM Orders
ORDER BY orderTime
LIMIT 3;
```

4.4.6 Top-N

功能描述

Top-N 查询是根据列排序找到N个最大或最小的值。最大值集和最小值集都被视为是一种 Top-N 的查询。若在批处理或流处理的表中需要显示出满足条件的 N 个最底层记录或最顶层记录， Top-N 查询将会十分有用。

语法格式

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     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]
```

语法说明

- ROW_NUMBER(): 根据当前分区内的各行的顺序从第一行开始，依次为每一行分配一个唯一且连续的号码。目前，我们只支持 ROW_NUMBER 在 over 窗口函数中使用。未来将会支持 RANK() 和 DENSE_RANK()函数。
- PARTITION BY col1[, col2...]: 指定分区列，每个分区都将会有一个 Top-N 结果。
- ORDER BY col1 [asc|desc][, col2 [asc|desc]...]: 指定排序列，不同列的排序方向可以不一样。
- WHERE rownum <= N: Flink 需要 rownum <= N 才能识别一个查询是否为 Top-N 查询。 其中， N 代表最大或最小的 N 条记录会被保留。
- [AND conditions]: 在 where 语句中，可以随意添加其他的查询条件，但其他条件只允许通过 AND 与 rownum <= N 结合使用。

 版权所有 © 华为技术有限公司

注意事项

- TopN 查询的结果会带有更新。
- Flink SQL 会根据排序键对输入的流进行排序。
- 如果 top N 的记录发生了变化，变化的部分会以撤销、更新记录的形式发送到下游。
- 如果 top N 记录需要存储到外部存储，则结果表需要拥有相同与 Top-N 查询相同的唯一键。

示例

查询每个分类实时销量最大的五个产品

```
SELECT *
  FROM (
     SELECT *,
         ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) as row_num
     FROM ShopSales)
  WHERE row_num <= 5;
```

4.4.7 去重

功能描述

对在列的集合内重复的行进行删除，只保留第一行或最后一行数据。

语法格式

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]
       ORDER BY time_attr [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1
```

语法说明

- ROW_NUMBER(): 从第一行开始，依次为每一行分配一个唯一且连续的号码。
- PARTITION BY col1[, col2...]: 指定分区的列，例如去重的键。
- ORDER BY time_attr [asc|desc]: 指定排序的列。所指定的列必须为时间属性。目前仅支持proctime。升序（ ASC ）排列指只保留第一行，而降序排列（ DESC ）则只保留最后一行。
- WHERE rownum = 1: Flink 需要 rownum = 1 以确定该查询是否为去重查询。

注意事项

无

示例

根据order_id对数据进行去重，其中proctime为事件时间属性列

```
SELECT order_id, user, product, number
  FROM (
     SELECT *,
```

 版权所有 © 华为技术有限公司

```
      ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY proctime ASC) as row_num
   FROM Orders)
WHERE row_num = 1;
```

4.5 函数

4.5.1 自定义函数

概述

DLI支持三种自定义函数：

- UDF：自定义函数，支持一个或多个输入参数，返回一个结果值。
- UDTF：自定义表值函数，支持一个或多个输入参数，可返回多行多列。
- UDAF：自定义聚合函数，将多条记录聚合成一个值。

POM 依赖

```
<dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-table-common</artifactId>
      <version>1.10.0</version>
      <scope>provided</scope>
</dependency>
```

注意事项

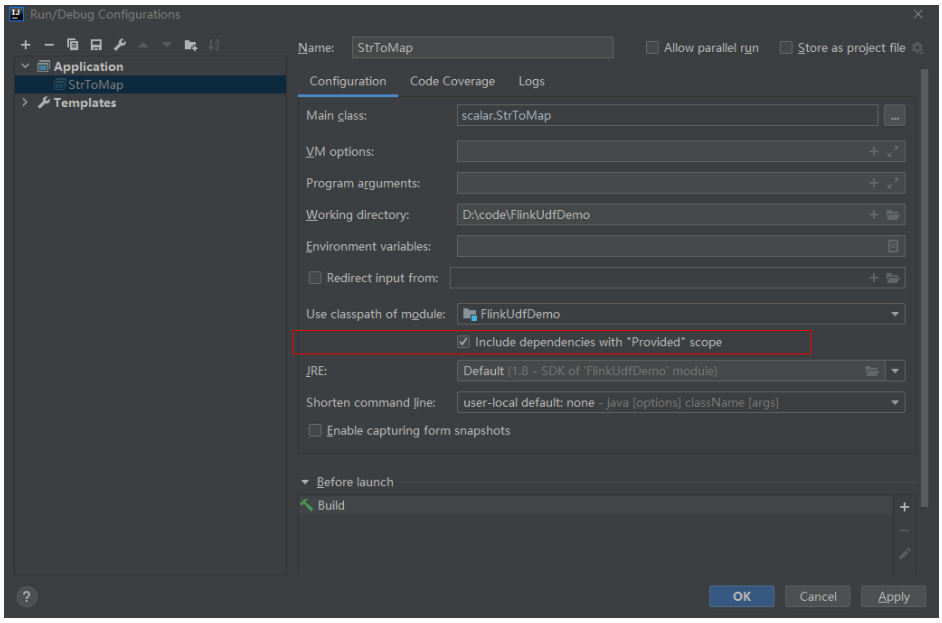
- 暂不支持通过python写UDF、UDTF、UDAF自定义函数。
- 如果使用IntelliJ IDEA工具对创建的自定义函数进行调试，则需要在IDEA上勾选：include dependencies with "Provided" scope，否则本地调试运行时会加载不到pom文件中的依赖包。

  具体操作以IntelliJ IDEA版本2020.2为例，参考如下：

  a. 在IntelliJ IDEA界面，选择调试的配置文件，单击“Edit Configurations”。

  b. 在“Run/Debug Configurations”界面，勾选：include dependencies with "Provided" scope。

版权所有 © 华为技术有限公司



c. 单击“OK”完成应用配置。

使用方式

1. 将写好的自定义函数打成JAR包，并上传到OBS上。
2. 在DLI管理控制台的左侧导航栏中，单击数据管理>“程序包管理”，然后点击创建，并使用OBS中的jar包创建相应的程序包。
3. 在DLI管理控制台的左侧导航栏中，单击作业管理>“Flink作业”，在需要编辑作业对应的“操作”列中，单击“编辑”，进入作业编辑页面。
4. 在“运行参数设置”页签，“UDF Jar”选择创建的程序包，单击“保存”。
5. 选定JAR包以后，SQL里添加UDF声明语句，就可以像普通函数一样使用了。

```
CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
```

UDF

UDF函数需继承ScalarFunction函数，并实现eval方法。open函数及close函数可选。

编写代码示例

```
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class UdfScalarFunction extends ScalarFunction {
  private int factor = 12;
  public UdfScalarFunction() {
    this.factor = 12;
  }
  /**
   * 初始化操作，可选
   * @param context
   */
  @Override
  public void open(FunctionContext context) {}
  /**
   * 自定义逻辑
   * @param s
   * @return
   */
  public int eval(String s) {
    return s.hashCode() * factor;
```


版权所有 © 华为技术有限公司

```
  }
  /**
   * 可选
   */
  @Override
  public void close() {}
}
```

使用示例

```
CREATE FUNCTION udf_test AS 'com.huaweicompany.udf.UdfScalarFunction';
INSERT INTO sink_stream select udf_test(attr) FROM source_stream;
```

UDTF

UDTF函数需继承TableFunction函数，并实现eval方法。open函数及close函数可选。如果需要UDTF返回多列，只需要将返回值声明成Tuple或Row即可。若使用Row，需要重载getResultType声明返回的字段类型。

编写代码示例

```
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class UdfTableFunction extends TableFunction<Row> {
  private Logger log = LoggerFactory.getLogger(TableFunction.class);
  /**
   * 初始化操作，可选
   * @param context
   */
  @Override
  public void open(FunctionContext context) {}
  public void eval(String str, String split) {
    for (String s : str.split(split)) {
      Row row = new Row(2);
      row.setField(0, s);
      row.setField(1, s.length());
      collect(row);
    }
  }
  /**
   * 函数返回类型声明
   * @return
   */
  @Override
  public TypeInformation<Row> getResultType() {
  return Types.ROW(Types.STRING, Types.INT);
  }
  /**
   * 可选
   */
  @Override
  public void close() {}
 }
```

使用示例

UDTF支持CROSS JOIN和LEFT JOIN，在使用UDTF时需要带上 LATERAL 和TABLE 两个关键字。

- CROSS JOIN：对于左表的每一行数据，假设UDTF不产生输出，则这一行不进行输出。

 版权所有 © 华为技术有限公司

- LEFT JOIN：对于左表的每一行数据，假设UDTF不产生输出，这一行仍会输出，UDTF相关字段用null填充。

```
CREATE FUNCTION udtf_test AS 'com.huaweicompany.udf.TableFunction';
// CROSS JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream, LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length);
// LEFT JOIN
INSERT INTO sink_stream select subValue, length FROM source_stream LEFT JOIN LATERAL
TABLE(udtf_test(attr, ',')) as T(subValue, length) ON TRUE;
```

UDAF

UDAF函数需继承AggregateFunction函数。首先需要创建一个用来存储计算结果的Accumulator，如示例里的WeightedAvgAccum。

编写代码示例

```
public class WeightedAvgAccum {
public long sum = 0;
public int count = 0;
}
```

```
import org.apache.flink.table.functions.AggregateFunction;
import java.util.Iterator;
/**
* 第一个类型变量为聚合函数返回的类型，第二个类型变量为Accumulator类型
* Weighted Average user-defined aggregate function.
*/
public class UdfAggFunction extends AggregateFunction<Long, WeightedAvgAccum> {
  // 初始化Accumulator
  @Override
  public WeightedAvgAccum createAccumulator() {
    return new WeightedAvgAccum();
  }
  // 返回Accumulator存储的中间计算值
  @Override
  public Long getValue(WeightedAvgAccum acc) {
    if (acc.count == 0) {
      return null;
    } else {
      return acc.sum / acc.count;
 }
}
// 根据输入更新中间计算值
public void accumulate(WeightedAvgAccum acc, long iValue) {
acc.sum += iValue;
acc.count += 1;
}
// Restract撤回操作，和accumulate操作相反
public void retract(WeightedAvgAccum acc, long iValue) {
acc.sum -= iValue;
acc.count -= 1;
}
// 合并多个accumulator值
public void merge(WeightedAvgAccum acc, Iterable<WeightedAvgAccum> it) {
Iterator<WeightedAvgAccum> iter = it.iterator();
while (iter.hasNext()) {
WeightedAvgAccum a = iter.next();
acc.count += a.count;
acc.sum += a.sum;
}
}
// 重置中间计算值
public void resetAccumulator(WeightedAvgAccum acc) {
acc.count = 0;
acc.sum = 0L;
```

 版权所有 © 华为技术有限公司

```
}
}
```

使用示例

```
CREATE FUNCTION udaf_test AS 'com.huaweicompany.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

4.5.2 内置函数

4.5.2.1 数学运算函数

关系运算符

所有数据类型都可用关系运算符进行比较，并返回一个BOOLEAN类型的值。

关系运算符均为双目操作符，被比较的两个数据类型必须是相同的数据类型或者是可以进行隐式转换的类型。

Flink SQL提供的关系运算符，请参见表4-30。

表 4-30 关系运算符

运算符	返回类型	描述
A = B	BOOLEAN	若A与B相等，返回TRUE，否则返回FALSE。用于做赋值操作。
A <> B	BOOLEAN	若A与B不相等，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL，该种运算符为标准SQL语法。
A < B	BOOLEAN	若A小于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A <= B	BOOLEAN	若A小于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A > B	BOOLEAN	若A大于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A >= B	BOOLEAN	若A大于或者等于B，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A IS NULL	BOOLEAN	若A为NULL则返回TRUE，否则返回FALSE。
A IS NOT NULL	BOOLEAN	若A不为NULL，则返回TRUE，否则返回FALSE。
A IS DISTINCT FROM B	BOOLEAN	若A与B不相等，则返回TRUE，将空值视为相同。
A IS NOT DISTINCT FROM B	BOOLEAN	若A与B相等，则返回TRUE，将空值视为相同。

版权所有 © 华为技术有限公司

运算符	返回类型	描述
A BETWEEN [ASYMMETRIC | SYMMETRIC] B AND C	BOOLEAN	若A大于或等于B且小于或等于C，则返回TRUE。 ● ASYMMETRIC：表示B和C位置相关。 例如：A BETWEEN ASYMMETRIC B AND C 等价于 (A BETWEEN B AND C)。 ● SYMMETRIC：表示B和C位置不相关。 例如：A BETWEEN SYMMETRIC B AND C 等价于 (A BETWEEN B AND C) OR (A BETWEEN C AND B)。
A NOT BETWEEN B [ASYMMETRIC | SYMMETRIC]AND C	BOOLEAN	若A小于B或大于C，则返回TRUE。 ● ASYMMETRIC：表示B和C位置相关。 例如：A NOT BETWEEN ASYMMETRIC B AND C 等价于 (A NOT BETWEEN B AND C)。 ● SYMMETRIC：表示B和C位置不相关。 例如：A NOT BETWEEN SYMMETRIC B AND C 等价于 (A NOT BETWEEN B AND C) OR (A NOT BETWEEN C AND B)。
A LIKE B [ESCAPE C]	BOOLEAN	若A与模式B匹配，则返回TRUE。 必要时可以定义转义字符C。
A NOT LIKE B [ESCAPE C]	BOOLEAN	若A与模式B不匹配，则返回TRUE。 必要时可以定义转义字符C。
A SIMILAR TO B [ESCAPE C]	BOOLEAN	若A与正则表达式B匹配，则返回TRUE。 必要时可以定义转义字符C。
A NOT SIMILAR TO B [ESCAPE C]	BOOLEAN	若A与正则表达式B不匹配，则返回TRUE。 必要时可以定义转义字符C。
value IN (value [, value]*)	BOOLEAN	若值等于列表中的值，则返回TRUE。
value NOT IN (value [, value]*)	BOOLEAN	若值不等于列表中的每个值，则返回TRUE。
EXISTS (sub-query)	BOOLEAN	若子查询至少返回一条数据，则返回TRUE。
value IN (sub-query)	BOOLEAN	若值等于子查询返回的某个值，则返回TRUE。
value NOT IN (sub-query)	BOOLEAN	若值不等于子查询返回的每个值，则返回TRUE。

注意事项

版权所有 © 华为技术有限公司

- double、real和float值存在一定的精度差。且我们不建议直接使用等号“=”对两个double类型数据进行比较。用户可以使用两个double类型相减，而后取绝对值的方式判断。当绝对值足够小时，认为两个double数值相等，例如:
  abs(0.9999999999 - 1.0000000000) < 0.000000001 //0.9999999999和1.0000000000为10位精度，而0.000000001为9位精度，此时可以认为0.9999999999和1.0000000000相等。
- 数值类型可与字符串类型进行比较。做大小(>,<,>=,<=)比较时，会默认将字符串转换为数值类型，因此不支持字符串内有除数字字符之外的字符。
- 字符串之间可以进行比较。

逻辑运算符

常用的逻辑操作符有AND、OR和NOT，优先级顺序为：NOT>AND>OR。

运算规则请参见表4-31，表中的A和B代表逻辑表达式。

表 4-31 逻辑运算符

运算符	返回类型	描述
A OR B	BOOLEAN	若A或B为TRUE，则返回TRUE，且支持三值逻辑。
A AND B	BOOLEAN	若A和B为TRUE，则返回TRUE，且支持三值逻辑。
NOT A	BOOLEAN	若A不为TRUE则返回TRUE；若A为UNKNOWN，返回UNKNOWN。
A IS FALSE	BOOLEAN	若A为FALSE则返回TRUE； 若A为UNKNOWN，则返回FALSE。
A IS NOT FALSE	BOOLEAN	若A不为FALSE则返回TRUE；若A为UNKNOWN，则返回TRUE。
A IS TRUE	BOOLEAN	若A为TRUE，则返回TRUE；若A为UNKNOWN，则返回FALSE。
A IS NOT TRUE	BOOLEAN	若A不为TRUE则返回TRUE；若A为UNKNOWN，则返回TRUE。
A IS UNKNOWN	BOOLEAN	若A为UNKNOWN，则返回TRUE。
A IS NOT UNKNOWN	BOOLEAN	若A不为UNKNOWN，则返回TRUE。

注意事项

逻辑操作符只允许boolean类型参与运算，不支持隐式类型转换。

算术运算符

算术运算符包括双目运算符与单目运算符，这些运算符都将返回数字类型。Flink SQL所支持的算术运算符如表4-32所示。

版权所有 © 华为技术有限公司

表 4-32 算术运算符

运算符	返回类型	描述
+ numeric	所有数字类型	返回数字。
- numeric	所有数字类型	返回负数。
A + B	所有数字类型	A和B相加。结果数据类型与操作数据类型相关，例如一个整数类型数据加上一个浮点类型数据，结果数值为浮点类型数据。
A - B	所有数字类型	A和B相减。结果数据类型与操作数据类型相关。
A * B	所有数字类型	A和B相乘。结果数据类型与操作数据类型相关。
A / B	所有数字类型	A和B相除。结果是一个double（双精度）类型的数值。
POWER(A, B)	所有数字类型	返回A数的B次方乘幂。
ABS(numeric)	所有数字类型	返回数值的绝对值。
MOD(A, B)	所有数字类型	返回A除以B的余数（模数）。 返回值只有在A为负数时才为负数。
SQRT(A)	所有数字类型	返回A的平方根。
LN(A)	所有数字类型	返回A的自然对数（基数e）。
LOG10(A)	所有数字类型	返回A的基数10对数。
LOG2(A)	所有数字类型	返回A的基数2对数。

版权所有 © 华为技术有限公司

运算符	返回类型	描述
LOG(B) LOG(A, B)	所有数字类型	当只有一个参数，返回B的自然对数（基数e）。 当有两个参数，返回B以A为基数的对数。 B必须大于0，且A必须大于1。
EXP(A)	所有数字类型	返回e的a次方。
CEIL(A) CEILING(A)	所有数字类型	将参数向上舍入为最接近的整数。例如ceil(21.2)，返回22。
FLOOR(A)	所有数字类型	对给定数据进行向下舍入最接近的整数。例如floor(21.2)，返回21。
SIN(A)	所有数字类型	计算给定A的正弦值。
COS(A)	所有数字类型	计算给定A的余弦值。
TAN(A)	所有数字类型	计算给定A的正切值。
COT(A)	所有数字类型	计算给定A的余切值。
ASIN(A)	所有数字类型	计算给定A的反正弦值。
ACOS(A)	所有数字类型	计算给定A的反余弦值。
ATAN(A)	所有数字类型	计算给定A的反正切值。
ATAN2(A, B)	所有数字类型	计算给定坐标(A, B)的反正切值。
COSH(A)	所有数字类型	计算给定A的双曲余弦值。返回类型为DOUBLE。

版权所有 © 华为技术有限公司

运算符	返回类型	描述
DEGREES(A)	所有数字类型	返回弧度所对应的角度。
RADIANS(A)	所有数字类型	返回角度所对应的弧度。
SIGN(A)	所有数字类型	返回a所对应的正负号，a为正返回1，a为负，返回-1，否则返回0。
ROUND(A, d)	所有数字类型	返回小数部分，d位之后数字的四舍五入，d为int型。例如round(21.263,2)，返回21.26。
PI	所有数字类型	返回pi的值。
E()	所有数字类型	返回e的值。
RAND()	所有数字类型	返回一个0.0和1.0之间的随机double类型的数（包含0.0，不包含1.0）。
RAND(A)	所有数字类型	根据初始化种子A，返回一个0.0和1.0之间的随机double类型的数（包含0.0，不包含1.0）。若初始化种子相同，则返回的随机数相同。
RAND_INTEGER(A)	所有数字类型	返回一个0和A之间的随机整数（包含0，不包含A）。
RAND_INTEGER(A, B)	所有数字类型	根据初始化种子A，返回一个0和B之间的随机整数值（包含0，不包含B）
UUID()	所有数字类型	返回一个UUID字符串。
BIN(A)	所有数字类型	返回一个整数A的二进制字符串。如为null则返回null。
HEX(A) HEX(B)	所有数字类型	返回一个整数A或者字符串B的十六进制字符串。若A或B为null，则返回null。

版权所有 © 华为技术有限公司

运算符	返回类型	描述
TRUNCATE(A, d)	所有数字类型	返回保留小数点后d为小数的数字。若A或d为null，则返回null。 例如：truncate(42.345, 2) = 42.340 truncate(42.345) = 42.000
PI()	所有数字类型	返回pi的值

注意事项

字符串类型不能参与算术运算。

4.5.2.2 字符串函数

字符串函数

数据湖探索（DLI）提供了丰富的字符串函数，用于处理和转换字符串数据。这些函数包括拼接、大小写转换、截取子串、替换、正则匹配、编码解码、格式转换等。此外，还支持字符串长度计算、位置查找、填充、反转等功能，以及从JSON字符串中提取值的JSON_VAL函数。这些功能广泛应用于数据清洗、文本处理和数据分析场景，为开发者提供强大的工具支持。

字符串函数简介请参考**表4-33**，更多内容参考**Apache Flink**。

表 4-33 字符串函数

函数	返回类型	描述
string1 || string2	STRING	返回两个字符串的拼接
CHAR_LENGTH(string) CHARACTER_LENGTH(string)	INT	返回字符串中的字符数量
UPPER(string)	STRING	返回字符串的大写形式
LOWER(string)	STRING	返回字符串的小写形式
POSITION(string1 IN string2)	INT	返回第一个字符串在第二个字符串中首次出现的位置。若第一个字符串不存在与第二个字符串，则返回0

版权所有 © 华为技术有限公司

函数	返回类型	描述
TRIM([BOTH | LEADING | TRAILING] string1 FROM string2)	STRING	去除string2字符串的首尾(或首部、或尾部)的string1字符串
LTRIM(string)	STRING	返回去除首部空格后的字符串 例如LTRIM(' This is a test String.') 返回"This is a test String."
RTRIM(string)	STRING	返回去除尾部空格后的字符串 例如RTRIM('This is a test String. ') 返回"This is a test String."
REPEAT(string, integer)	STRING	返回integer个string连接后的字符串 例如REPEAT('This is a test String.', 2) 返回"This is a test String.This is a test String."
REGEXP_REPLACE(string1, string2, string3)	STRING	用string3代替string1中的符合正则表达式string2的字符串，并返回替换后的string1字符串 例如REGEXP_REPLACE('foobar', 'oo|ar', '') 返回 "fb" REGEXP_REPLACE('ab\ab', '\\', 'e')返回"abeab"
OVERLAY(string1 PLACING string2 FROM integer1 [FOR integer2])	STRING	用string2代替string1中的字符串，从integer1开始，替换长度为integer2，并返回替换后的string1字符串 integer2默认为string2的长度 例如OVERLAY('This is an old string' PLACING ' new' FROM 10 FOR 5)返回"This is a new string"
SUBSTRING(string FROM integer1 [FOR integer2])	STRING	返回string中从integer1位置开始的长度为integer2的子字符串。若integer2未配置，则默认返回从integer1开始到末尾的子字符串
REPLACE(string1, string2, string3)	STRING	用string3代替string1中的string2后的字符串，并返回替换后的string1字符串 例如：REPLACE('hello world', 'world', 'flink') 返回 "hello flink" REPLACE('abababb', 'abab', 'z') 返回 "zab" REPLACE('ab\\ab', '\\', 'e')返回"abeab"
REGEXP_EXTRACT(string1, string2[, integer])	STRING	使用正则表达式string2匹配抽取字符串string1中的第integer个字串，integer从1开始，正则匹配提取。 若参数为 NULL或者正则不合法，则返回NULL。 例如REGEXP_EXTRACT('foothebar', 'foo(.*?)(bar)', 2)" 返回"bar"

版权所有 © 华为技术有限公司

函数	返回类型	描述
INITCAP(string)	STRING	返回将字符串的首字符大写其余字符转为小写后的字符串
CONCAT(string1, string2,...)	STRING	返回将两个或多个字符串拼接后的新字符串。 例如 CONCAT('AA', 'BB', 'CC') 返回"AABBCC"
CONCAT_WS(string1, string2, string3,...)	STRING	返回将每个参数和第一个参数指定的分隔符依次连接到一起组成的字符串。若string1是null，则返回null。若其他参数为null，在执行拼接过程中跳过取值为null的参数 例如CONCAT_WS('~', 'AA', NULL, 'BB', '', 'CC') 返回"AA~BB~~CC"
LPAD(string1, integer, string2)	STRING	将string2字符串拼接到string1字符串的左端，直到新的字符串达到指定长度integer为止 任意参数为null时，返回null 若integer为负数，则返回null 若integer不大于string1的长度，则返回string1裁剪为integer长度的字符串 例如LPAD('hi',4,'??') 返回"??hi" LPAD('hi',1,'??') 返回"h"
RPAD(string1, integer, string2)	STRING	将string2字符串拼接到string1字符串的右端，直到新的字符串达到指定长度integer为止 任意参数为null时，返回null 若integer为负数，则返回null 若integer不大于string1的长度，则返回string1裁剪为integer长度的字符串 例如RPAD('hi',4,'??') 返回 "hi??" RPAD('hi',1,'??') 返回"h"
FROM_BASE64(string)	STRING	将base64编码的字符串str解析成对应字符串 若字符串为null，则返回null 例如FROM_BASE64('aGVsbG8gd29ybGQ=') 返回"hello world"
TO_BASE64(string)	STRING	将字符串基于base64编码 若字符串为null，则返回null 例如TO_BASE64('hello world') 返回"aGVsbG8gd29ybGQ="
ASCII(string)	INT	返回字符串的第一个字符的ASCII值 若字符串为null，则返回null 例如ascii('abc') 返回97 ascii(CAST(NULL AS VARCHAR)) 返回NULL

版权所有 © 华为技术有限公司

函数	返回类型	描述
CHR(integer)	STRING	将ASCII码转换为字符 若integer大于255，则计算出integer除以255的余数，并将余数作为ASCII码值 若integer为null，则返回null chr(97) 返回a chr(353) 返回a
DECODE(binary, string)	STRING	使用提供的字符集string解码参数binary，字符集可以为'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16' 若任意参数为null，则返回null
ENCODE(strinh1, string2)	STRING	使用提供的字符集string2编码字符串string1，字符集可以为'US-ASCII', 'ISO-8859-1', 'UTF-8', 'UTF-16BE', 'UTF-16LE', 'UTF-16' 若任意参数为null，则返回null
INSTR(string1, string2)	INT	返回string2在string1中首次出现的位置 若有参数为null，则返回null
LEFT(string, integer)	STRING	返回最左边的integer个字符 若integer为负数，则返回空 若存在参数为null，则返回null
RIGHT(string, integer)	STRING	返回最右侧的integer个字符 若integer为负数，则返回空 若存在参数为null，则返回null
LOCATE(string1, string2[, integer])	INT	返回string1在string2的位置integer之后首次出现的位置 若string1在string2的位置integer之后不存在，则返回0 若integer不存在，则默认为0 若存在参数为null，则返回null

版权所有 © 华为技术有限公司

函数	返回类型	描述
PARSE_URL(string1, string2[, string3])	STRING	返回URL string1中指定的部分解析后的值 string2为'HOST'、'PATH'、'QUERY'、'REF'、'PROTOCOL'、'AUTHORITY'、'FILE'或'USERINFO' 若存在参数为null，则返回null 若string2为QUERY，也可以指定QUERY中的key为string3 例如： parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST')返回 'facebook.com' parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') 返回'v1'
REGEXP(string1, string2)	BOOLEAN	对指定的字符串执行一个正则表达式搜索，并返回一个BOOLEAN值表示是否找到指定的匹配模式。若找到，则返回TRUE。其中string1表示指定的字符串，string2表示正则表达式 若存在参数为null，则返回null
REVERSE(string)	STRING	反转字符串，返回字符串值的相反顺序。 若存在参数为null，则返回null
SPLIT_INDEX(string1, string2, integer1)	STRING	以string2作为分隔符，将字符串string1分割成若干段，取其中的第integer1段。integer1从0开始 若integer1为负数，则返回null 如果任一参数为null，则返回null
STR_TO_MAP(string1[, string2, string3]])	MAP	使用string2分隔符将string1分割成K-V对，并使用string3分隔每个K-V对，组装成MAP返回 string2默认为',' string3默认为'='
SUBSTR(string[, integer1[, integer2]])	STRING	截取从位置integer1开始，长度为integer2的子串，并返回 若为指定integer2，翻截取到字符串结尾
JSON_VAL(STRING json_string, STRING json_path)	STRING	从json形式的字符串json_string中提取指定json_path的值。具体函数使用可以参考**JSON_VAL函数使用说明**说明。 **说明** 以下规则优先级按照顺序从高到低。 1. 不允许json_string和json_path为NULL 2. json_string格式必须为合法的json串，否则函数返回NULL 3. json_string为空字符串，则函数返回空字符串 4. json_path为空字串或路径不存在，则函数返回NULL

版权所有 © 华为技术有限公司

JSON_VAL 函数使用说明

- **语法**

```
STRING JSON_VAL(STRING json_string, STRING json_path)
```

表 4-34 参数说明

参数	数据类型	说明
json_string	STRING	需要解析的JSON对象，使用字符串表示。
json_path	STRING	解析JSON的路径表达式，使用字符串表示。 目前path支持如下表达式参考下表**表4-35**。

表 4-35 json_path 参数支持的表达式

表达式	说明
$	根对象
[]	数组下标
*	数组通配符
.	取子元素

- **示例**

a. 测试输入数据。

测试数据源kafka，具体消息内容参考如下：

```
"{name:James,age:24,gender:male,grade:{math:95,science:[80,85],english:100}}"
"{name:James,age:24,gender:male,grade:{math:95,science:[80,85],english:100}]"
```

b. 使用JSON_VAL编写SQL

```
create table kafkaSource(
  message STRING
)
with (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = '*topic-swq*',
  'connector.properties.bootstrap.servers' =
'xxx.xxx.xxx.xxx:9092,yyy.yyy.yyy:9092,zzz.zzz.zzz.zzz:9092',
  'connector.startup-mode' = 'earliest-offset',
  'format.field-delimiter' = '|',
  'format.type' = 'csv'
);

create table kafkaSink(
  message1 STRING,
  message2 STRING,
  message3 STRING,
  message4 STRING,
  message5 STRING,
  message6 STRING
)
with (
  'connector.type' = 'kafka',
```


版权所有 © 华为技术有限公司

```
  'connector.version' = '0.11',
  'connector.topic' = 'topic-swq-out',
  'connector.properties.bootstrap.servers' =
'xxx.xxx.xxx.xxx:9092,yyy.yyy.yyy:9092,zzz.zzz.zzz.zzz:9092',
  'format.type' = 'json'
);

INSERT INTO kafkaSink
SELECT
JSON_VAL(message,""),
JSON_VAL(message,"$.name"),
JSON_VAL(message,"$.grade.science"),
JSON_VAL(message,"$.grade.science[*]"),
JSON_VAL(message,"$.grade.science[1]"),
JSON_VAL(message,"$.grade.dddd")
FROM kafkaSource;
```

c. 查看输出结果

```
{"message1":null,"message2":"swq","message3":"[80,85]","message4":"[80,85]","message5":"85"
,"message6":null}
{"message1":null,"message2":null,"message3":null,"message4":null,"message5":null,"message6":
null}
```

4.5.2.3 时间函数

Flink OpenSource SQL所支持的时间函数如表4-36所示。

函数说明

表 4-36 时间函数

函数	返回值	描述
DATE string	DATE	将日期字符串以**"yyyy-MM-dd"**的形式解析为SQL日期。
TIME string	TIME	将时间字符串以**"HH:mm:ss[.fff]"**形式解析为SQL时间。
TIMESTAMP string	TIMESTAMP	将时间字符串转换为时间戳，时间字符串格式为："**yyyy-MM-dd HH:mm:ss[.fff]**"。

 版权所有 © 华为技术有限公司

函数	返回值	描述
INTERVAL string range	INTERVAL	interval表示时间间隔。有两种类型，分别为： • 一种为"yyyy-MM"即保存年份和月份，精度到月份，它的range参数可以为YEAR或者YEAR To Month。 • 一种为天时间"dd HH:mm:sss.fff"，用来保存天数、小时、分钟、秒和毫秒， 精度最低到毫秒。它的range参数可以为DAY、MINUTE、DAY TO HOUR、DAY TO SECOND。 例如： INTERVAL '10 00:00:00.004' DAY TO second表示间隔10天4毫秒。 INTERVAL '10' DAY表示间隔10天 INTERVAL '2-10' YEAR TO MONTH表示间隔2年10个月。
CURRENT_DATE	DATE	以UTC时区返回当前SQL日期。
CURRENT_TIME	TIME	以UTC时区返回当前SQL时间。
CURRENT_TIMESTAMP	TIMESTAMP	以UTC时区返回当前SQL时间戳。
LOCALTIME	TIME	返回当前时区的当前SQL时间。
LOCALTIMESTAMP	TIMESTAMP	返回当前时区的当前SQL时间戳。
EXTRACT(timeintervalunit FROM temporal)	BIGINT	提取时间点的一部分或者时间间隔。以int类型返回该部分。 例如：提取日期“2006-06-05”中的日为5 EXTRACT(DAY FROM DATE '2006-06-05') 返回5。
YEAR(date)	BIGINT	返回输入时间的年份 例如：YEAR(DATE '1994-09-27') 返回1994
QUARTER(date)	BIGINT	从SQL日期返回表示该日期季度的数字。
MONTH(date)	BIGINT	返回输入时间的月份 例如：MONTH(DATE '1994-09-27')返回9
WEEK(date)	BIGINT	计算当前日期是一年中的第几周 例如：WEEK(DATE '1994-09-27') 返回39
DAYOFYEAR(date)	BIGINT	计算当前日期是一年中的第几天 例如：DAYOFYEAR(DATE '1994-09-27') 返回270

版权所有 © 华为技术有限公司

函数	返回值	描述
DAYOFMONTH(date)	BIGINT	计算当前日期是这个月的第几天 例如：DAYOFMONTH(DATE '1994-09-27') 返回27
DAYOFWEEK(date)	BIGINT	计算当前日期是当前周的第几天 其中周日设为1 例如：DAYOFWEEK(DATE '1994-09-27') 返回3
HOUR(timestamp)	BIGINT	返回当前时间戳的24小时制的小时数，范围0-23 例如：HOUR(TIMESTAMP '1994-09-27 13:14:15') 返回13
MINUTE(timestamp)	BIGINT	返回当前时间戳中的分钟数，范围0-59 例如：MINUTE(TIMESTAMP '1994-09-27 13:14:15') 返回14
SECOND(timestamp)	BIGINT	返回当前时间戳中的秒数，范围0-59 例如：SECOND(TIMESTAMP '1994-09-27 13:14:15') 返回15
FLOOR(timepoint TO timeintervalunit)	TIME	向下对齐时间。 例如：FLOOR(TIME '12:44:31' TO MINUTE) 按分钟对齐到12:44:00。
CEIL(timepoint TO timeintervalunit)	TIME	向上对齐时间。 例如：CEIL(TIME '12:44:31' TO MINUTE)按分钟对齐到12:45:00。
(timepoint1, temporal1) OVERLAPS (timepoint2, temporal2)	BOOLEAN	若两个时间范围有重叠，则返回TRUE 例如： (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) 返回TRUE (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) 返回FALSE
DATE_FORMAT(timestamp, string)	STRING	将日期从源格式转换至目标格式
TIMESTAMPADD(timeintervalunit, interval, timepoint)	TIMESTAMP/ DATE/ TIME	将整型interval与timeintervalunit组成的结果添加日期或日期时间到timepoint中，并返回添加后的日期时间 例如：TIMESTAMPADD(WEEK, 1, DATE '2003-01-02') 返回2003-01-09

版权所有 © 华为技术有限公司

函数	返回值	描述
TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)	INT	返回timepoint1和timepoint2相差的时间单元数量 timepointunit表示时间单元，应该是SECOND、MINUTE、HOUR、DAY、MONTH或YEAR 例如： TIMESTAMPDIFF(DAY, TIMESTAMP '2003-01-02 10:00:00', TIMESTAMP '2003-01-03 10:00:00') 返回1
CONVERT_TZ(string1, string2, string3)	TIMESTAMP	将string2时区的时间string1转换为其在string3时区的对应时间 例如：CONVERT_TZ('1970-01-01 00:00:00', 'UTC', 'Country A/City A') 返回'1969-12-31 16:00:00'
FROM_UNIXTIME(numeric[, string])	STRING	根据时间戳numeric和当前时区返回string格式的时间 string默认格式为'YYYY-MM-DD hh:mm:ss' 例如：FROM_UNIXTIME(44)返回1970-01-01 09:00:44
UNIX_TIMESTAMP()	BIGINT	返回当前时间的时间戳，单位为秒
UNIX_TIMESTAMP(string1[, string2])	BIGINT	将string2格式的时间字符串string1转为时间戳，单位为秒 string2默认格式为'yyyy-MM-dd HH:mm:ss'
TO_DATE(string1[, string2])	DATE	将string2格式的日期字符串，转换为DATE类型 string2默认格式为 'yyyy-MM-dd'
TO_TIMESTAMP(string1[, string2])	TIMESTAMP	将string2格式的日期时间字符串转换为TIMESTAMP类型 string2默认格式为'yyyy-MM-dd HH:mm:ss'

DATE

- **功能描述**

  DATE函数将**"yyyy-MM-dd"**日期格式的字符串解析为DATE类型的日期。

- **语法说明**

  ```
  DATE DATE string
  ```

- **入参说明**

版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
string	STRING	SQL日期格式的字符串。 注意该字符串的格式必须为"yyyy-MM-dd"格式，否则语义校验会报错。

- **示例**
  - 测试语句

```
SELECT
    DATE "2021-08-19" AS `result`
FROM
    testtable;
```

  - 测试结果

result
2021-08-19

TIME

- **功能描述**

  将时间字符串以**"HH:mm:ss[.fff]"**形式解析为SQL时间，结果以TIME类型返回。

- **语法说明**

```
TIME TIME string
```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间字符串。 注意该字符串格式必须**"HH:mm:ss[.fff]"**，否则语义校验会报错。

- **示例**
  - 测试语句

```
SELECT
    TIME "10:11:12" AS `result`,
        TIME "10:11:12.032" AS `result2`
FROM
    testtable;
```

  - 测试结果

result	result2
10:11:12	10:11:12.032

 版权所有 © 华为技术有限公司

TIMESTAMP

- **功能描述**

  将时间字符串转换为时间戳，时间字符串格式为："**yyyy-MM-dd HH:mm:ss[.fff]**"，以TIMESTAMP(3)类型返回。

- **语法说明**

  ```
  TIMESTAMP(3) TIMESTAMP string
  ```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间戳字符串。 注意该字符串格式必须为"**yyyy-MM-dd HH:mm:ss[.fff]**"，否则语义校验会报错。

- **示例**

  - 测试语句

    ```
    SELECT
       TIMESTAMP "1997-04-25 13:14:15" AS `result`,
           TIMESTAMP "1997-04-25 13:14:15.032" AS `result2`
    FROM
       testtable;
    ```

  - 测试结果

result	result2
1997-04-25 13:14:15	1997-04-25 13:14:15.032

INTERVAL

- **功能描述**

  INTERVAL函数用于表示时间间隔。

- **语法说明**

  ```
  INTERVAL INTERVAL string range
  ```

- **入参说明**

参数名	数据类型	参数说明
string	STRING	时间戳字符串，搭配参数range使用。两种格式类型，分别为： • 一种为**"yyyy-MM"**即保存年份和月份，精度到月份，它的range参数可以为**YEAR**或者**YEAR To Month**。 • 一种为天时间**"dd HH:mm:sss.fff"**，用来保存天数、小时、分钟、秒和毫秒， 精度最低到毫秒。它的range参数可以为**DAY**、**MINUTE**、**DAY TO HOUR**、**DAY TO SECOND**。

版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
range	INTERVAL	时间间隔说明，搭配string参数使用，详细请参考string参数说明。 取值范围为：**YEAR、YEAR To Month、DAY、MINUTE、DAY TO HOUR、DAY TO SECOND**。

- **示例**

  测试语句

```
--表示间隔10天4毫秒。
INTERVAL '10 00:00:00.004' DAY TO second
--DAY表示间隔10天
INTERVAL '10'
--表示间隔2年10个月
INTERVAL '2-10' YEAR TO MONTH
```

CURRENT_DATE

- **功能描述**

  以UTC时区**"yyyy-MM-dd"**格式返回当前SQL日期，返回类型为DATE。

- **语法说明**

```
DATE CURRENT_DATE
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  CURRENT_DATE AS `result`
FROM
  testtable;
```

  - 测试结果

result
2021-10-28

CURRENT_TIME

- **功能描述**

  以UTC（UTC+0）时区“HH:mm:sss.fff”格式返回当前SQL时间，返回类型为TIME。

- **语法说明**

```
TIME CURRENT_TIME
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
  CURRENT_TIME AS `result`
```

 版权所有 © 华为技术有限公司

```
FROM
  testtable;
```

- 测试结果

result
08:29:19.289

CURRENT_TIMESTAMP

- **功能描述**

  以UTC（UTC+0）时区返回当前SQL时间戳，返回类型为TIMESTAMP(3)。

- **语法说明**

```
TIMESTAMP(3) CURRENT_TIMESTAMP
```

- **入参说明**

  无。

- **示例**
  - 测试语句

```
SELECT
  CURRENT_TIMESTAMP AS `result`
FROM
  testtable;
```

  - 测试结果

result
2021-10-28 08:33:51.606

LOCALTIME

- **功能描述**

  返回当前时区的当前SQL时间，返回类型为TIME。

- **语法说明**

```
TIME LOCALTIME
```

- **入参说明**

  无。

- **示例**
  - 测试语句

```
SELECT
  LOCALTIME AS `result`
FROM
  testtable;
```

  - 测试结果

result
16:39:37.706

 版权所有 © 华为技术有限公司

LOCALTIMESTAMP

- **功能描述**

  返回当前时区的当前SQL时间戳，返回类型为TIMESTAMP(3)。

- **语法说明**

```
TIMESTAMP(3) LOCALTIMESTAMP
```

- **入参说明**

  无。

- **示例**

  - 测试语句

```
SELECT
    LOCALTIMESTAMP AS `result`
FROM
    testtable;
```

  - 测试结果

result
2021-10-28 16:43:17.625

EXTRACT

- **功能描述**

  提取时间点或时间间隔中指定某一时间单位的部分，以BIGINT类型返回。

- **语法说明**

```
BIGINT EXTRACT(timeinteravlunit FROM temporal)
```

- **入参说明**

参数名	数据类型	参数说明
timeinteravlunit	TIMEUNIT	需要从时间点或时间间隔中提取的时间单位，取值可以是：YEAR/QUARTER/MONTH/WEEK/DAY/DOY/HOUR/MINUTE/SECOND。
temporal	DATE/TIME/TIMESTAMP/INTERVAL	时间点或时间间隔。

⚠ 注意

不允许指定不存在于时间点或时间间隔中的时间单位，否则作业会提交失败。

例如如下错误语句，会报错YEAR不能从TIME中提取。

```
SELECT
    EXTRACT(YEAR FROM TIME '12:44:31' ) AS `result`
FROM
    testtable;
```

- **示例**

版权所有 © 华为技术有限公司

- 测试语句

```
SELECT
    EXTRACT(YEAR FROM DATE '1997-04-25' ) AS `result`,
        EXTRACT(MINUTE FROM TIME '12:44:31') AS `result2`,
        EXTRACT(SECOND FROM TIMESTAMP '1997-04-25 13:14:15') AS `result3`,
        EXTRACT(YEAR FROM INTERVAL '2-10' YEAR TO MONTH) AS `result4`,
FROM
    testtable;
```

- 测试结果

result	result2	result3	result4
1997	44	15	2

YEAR

- **功能描述**

  从SQL日期date返回年份，以BIGINT类型返回。

- **语法说明**

```
BIGINT YEAR(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	DATE类型的SQL日期。

- **示例**
  - 测试语句

```
SELECT
    YEAR(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

  - 测试结果

result
1997

QUARTER

- **功能描述**

  从SQL日期返回表示该日期季度的数字（1到4之间的整数），返回类型为BIGINT。

- **语法说明**

```
BIGINT QUARTER(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

 版权所有 © 华为技术有限公司

- **示例**
  - 测试语句
    ```
    SELECT
      QUARTER(DATE '1997-04-25' ) AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
2

MONTH

- **功能描述**

  返回输入时间的月份（1到12之间的整数），返回类型为BIGINT。
- **语法说明**
  ```
  BIGINT MONTH(date)
  ```
- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**
  - 测试语句
    ```
    SELECT
      MONTH(DATE '1997-04-25' ) AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
4

WEEK

- **功能描述**

  计算当前日期是一年中的第几周，以BIGINT类型返回。
- **语法说明**
  ```
  BIGINT WEEK(date)
  ```
- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**

 版权所有 © 华为技术有限公司

- 测试语句

```
SELECT
  WEEK(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

- 测试结果

result
17

DAYOFYEAR

- **功能描述**

  计算当前日期是一年中的第几天（返回1到366 之间的整数），以BIGINT类型返回。

- **语法说明**

```
BIGINT DAYOFYEAR(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**

  - 测试语句

```
SELECT
  DAYOFYEAR(DATE '1997-04-25' ) AS `result`
FROM
  testtable;
```

  - 测试结果

result
115

DAYOFMONTH

- **功能描述**

  计算当前日期是这个月的第几天（1到31之间的整数），以BIGINT类型返回。

- **语法说明**

```
BIGINT DAYOFMONTH(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**

 版权所有 © 华为技术有限公司

- 测试语句

```
SELECT
    DAYOFMONTH(DATE '1997-04-25' ) AS `result`
FROM
    testtable;
```

- 测试结果

result
25

DAYOFWEEK

- **功能描述**

  计算当前日期是当前周的第几天（1 到 7之间的整数），以BIGINT类型返回。

  说明

  需要注意这里自然周的起点是星期天，即每周的第1天是星期天，第2天是星期一，以此类推。

- **语法说明**

```
BIGINT DAYOFWEEK(date)
```

- **入参说明**

参数名	数据类型	参数说明
date	DATE	SQL日期。

- **示例**

  - 测试语句

```
SELECT
    DAYOFWEEK(DATE '1997-04-25') AS `result`
FROM
    testtable;
```

  - 测试结果

result
6

HOUR

- **功能描述**

  从当前时间戳获取以24小时制的小时数进行返回，范围0-23（0 到 23 之间的整数），返回类型为BIGINT。

- **语法说明**

```
BIGINT HOUR(timestamp)
```

- **入参说明**

 版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
      HOUR(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
10

MINUTE

- **功能描述**

  返回当前时间戳中的分钟数（0 到 59 之间的整数），返回类型为BIGINT。
- **语法说明**
  ```
  BIGINT MINUTE(timestamp)
  ```
- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句
    ```
    SELECT
      MINUTE(TIMESTAMP '1997-04-25 10:11:12') AS `result`
    FROM
      testtable;
    ```
  - 测试结果

result
11

SECOND

- **功能描述**

  返回当前时间戳中的秒数（0 到 59 之间的整数），返回类型为BIGINT。
- **语法说明**
  ```
  BIGINT SECOND(timestamp)
  ```
- **入参说明**

 版权所有 © 华为技术有限公司

参数名	数据类型	参数说明
timestamp	TIMESTAMP	SQL时间戳。

- **示例**
  - 测试语句

```
SELECT
    SECOND(TIMESTAMP '1997-04-25 10:11:12') AS `result`
FROM
    testtable;
```

  - 测试结果

result
12

FLOOR

- **功能描述**

  返回将时间点向下取值到指定时间单位的值。

- **语法说明**

```
TIME/TIMESTAMP(3) FLOOR(timepoint TO timeintervalunit)
```

- **入参说明**

参数名	数据类型	参数说明
timepoint	TIMESTAMP /TIME	SQL时间或SQL时间戳。
timeintervalunit	TIMEUNIT	时间单位，类型可以是YEAR/QUARTER/ MONTH/WEEK/DAY/DOY/HOUR/MINUTE/ SECOND。

- **示例**
  - 测试语句。注意以下userDefined结果表语法说明，请参考**userDefined结果表**。

```
create table PrintSink (
  message TIME,
  message2 TIME,
  message3 TIMESTAMP(3)
)
with (
  'connector.type' = 'user-defined',
  'connector.class-name' = 'com.swqtest.flink.sink.PrintSink'--注意修改为自定义的类，具体请参考userDefined结果表语法说明。
);

INSERT INTO
    PrintSink
SELECT
    FLOOR(TIME '13:14:15' TO MINUTE) AS `result`
        FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
        FLOOR(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`;
```

  - 测试结果

 版权所有 © 华为技术有限公司

PrintSink表的字段值分别为：

message	message2	message3
13:14	13:14	1997-04-25T13:14

CEIL

- **功能描述**

  返回将时间点向上取值到指定时间单位的值。

- **语法说明**

  ```
  TIME/TIMESTAMP(3) CEIL(timepoint TO timeintervalunit)
  ```

- **入参说明**

参数名	数据类型	参数说明
timepoint	TIMESTAMP/TIME	SQL时间或SQL时间戳。
timeintervalunit	TIMEUNIT	时间单位，类型可以是YEAR/QUARTER/MONTH/WEEK/DAY/DOY/HOUR/MINUTE/SECOND。

- **示例**

  - 测试语句。注意以下userDefined结果表语法说明，请参考**userDefined结果表**。

    ```
    create table PrintSink (
      message TIME,
      message2 TIME,
      message3 TIMESTAMP(3)
    )
    with (
      'connector.type' = 'user-defined',
      'connector.class-name' = 'com.swqtest.flink.sink.PrintSink'--注意修改为自定义的类，具体请参考userDefined结果表语法说明。
    );

    INSERT INTO
        PrintSink
    SELECT
        CEIL(TIME '13:14:15' TO MINUTE) AS `result`
            CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result2`,
            CEIL(TIMESTAMP '1997-04-25 13:14:15' TO MINUTE) AS `result3`;
    ```

  - 测试结果

result	result2	result3
13:15	13:15	1997-04-25T13:15

OVERLAPS

- **功能描述**

  若两个时间范围有重叠，则返回TRUE，反之，则返回FALSE。

版权所有 © 华为技术有限公司

- **语法说明**
  BOOLEAN (timepoint1, temporal1) **OVERLAPS** (timepoint2, temporal2)
- **入参说明**

参数名	数据类型	参数说明
timepoint1/ timepoint2	DATE/TIME/ TIMESTAMP	时间点。
temporal1/ temporal2	DATE/TIME/ TIMESTAMP/ INTERVAL	时间点或时间间隔。

  说明
  - (timepoint, temporal)在判断是否重叠时为闭区间。
  - temporal可以是DATE/TIME/TIMESTAMP也可以是INTERVAL。
    - 当temporal是DATE/TIME/TIMESTAMP时，(timepoint, temporal)表示timepoint, temporal之间的时间间隔。允许temporal在timepoint之前，如(DATE '1997-04-25', DATE '1997-04-23')也合法。
    - 当temporal是INTERVAL时，(timepoint, temporal)表示timepoint, timepoint +temporal之间的时间间隔。
  - 必须保证(timepoint1, temporal1)和(timepoint2, temporal2)是同一数据类型的时间间隔。
- **示例**
  - 测试语句

```
SELECT
    (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS `result`,
        (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) AS `result2`,
    (TIME '2:30:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:31:00', INTERVAL '2' HOUR) AS `result3`,
    (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:00:00', INTERVAL '3' HOUR) AS `result4`,
    (TIMESTAMP '1997-04-25 12:00:00', TIMESTAMP '1997-04-25 12:20:00') OVERLAPS (TIMESTAMP '1997-04-25 13:00:00', INTERVAL '2' HOUR) AS `result5`,
    (DATE '1997-04-23', INTERVAL '2' DAY) OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY) AS `result6`,
    (DATE '1997-04-25', DATE '1997-04-23') OVERLAPS (DATE '1997-04-25', INTERVAL '2' DAY) AS `result7`
FROM
    testtable;
```

  - 测试结果

result	result2	result3	result4	result5	result6	result7
true	true	false	true	false	true	true

 版权所有 © 华为技术有限公司

DATE_FORMAT

- **功能描述**

  将时间戳或时间戳格式的字符串转换为指定格式的日期字符串。

- **语法说明**

  STRING **DATE_FORMAT**(timestamp, dateformat)

- **入参说明**

参数名	数据类型	参数说明
timestamp	TIMESTAMP/ STRING	时间点。
dateformat	STRING	日期格式字符串。

- **示例**

  - 测试语句

```
SELECT
    DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd HH:mm:ss') AS `result`,
        DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result2`,
    DATE_FORMAT(TIMESTAMP '1997-04-25 10:11:12', 'yy/MM/dd HH:mm') AS `result3`,
        DATE_FORMAT('1997-04-25 10:11:12', 'yyyy-MM-dd') AS `result4`
FROM    testtable;
```

  - 测试结果

result	result2	result3	result4
1997-04-25 10:11:12	1997-04-25	97/04/25 10:11	1997-04-25

TIMESTAMPADD

- **功能描述**

  参考语法说明，本函数功能为将整型interval与timeintervalunit组成的结果添加到timepoint中，并返回添加后的日期时间。

  说明

  TIMESTAMPADD函数返回结果与timepoint相同。例外场景为：如果timepoint输入类型为TIMESTAMP，也可以将TIMESTAMPADD函数返回结果插入到DATE类型的表字段中。

- **语法说明**

  TIMESTAMP(3)/DATE/TIME **TIMESTAMPADD**(timeintervalunit, interval, timepoint)

- **入参说明**

参数名	数据类型	参数说明
timeintervalunit	TIMEUNIT	时间单位。
interval	INT	整型的时间间隔。
timepoint	TIMESTAMP/ DATE/TIME	时间点

 版权所有 © 华为技术有限公司

- **示例**
  - 测试语句

```
SELECT
    TIMESTAMPADD(WEEK, 1, DATE '1997-04-25') AS `result`,
        TIMESTAMPADD(QUARTER, 1, TIMESTAMP '1997-04-25 10:11:12') AS `result2`,
    TIMESTAMPADD(SECOND, 2, TIME '10:11:12') AS `result3`
FROM    testtable;
```

  - 测试结果

result	result2	result3
1997-05-02	• 如果该字段插入到TIMESTAMP类型的表字段中，则返回：1997-07-25T10:11:12 • 如果该字段插入到DATE类型的表字段中，则返回：1997-07-25	10:11:14

TIMESTAMPDIFF

- **功能描述**

  参考语法说明，本函数功能为返回timepoint1和timepoint2之间的时间间隔，间隔的单位由第一个参数timepointunit指定。

- **语法说明**

```
INT TIMESTAMPDIFF(timepointunit, timepoint1, timepoint2)
```

- **入参说明**

参数名	数据类型	参数说明
timepointunit	TIMEUNIT	时间单位。取值范围为： SECOND、MINUTE、HOUR、DAY、MONTH、YEAR。
timepoint1/ timepoint2	TIMESTAMP/ DATE	时间点。

- **示例**
  - 测试语句

```
SELECT
    TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-25 10:00:00', TIMESTAMP '1997-04-28 10:00:00')
AS `result`,
        TIMESTAMPDIFF(DAY, DATE '1997-04-25', DATE '1997-04-28') AS `result2`,
    TIMESTAMPDIFF(DAY, TIMESTAMP '1997-04-27 10:00:20', TIMESTAMP '1997-04-25 10:00:00')
AS `result3`
FROM    testtable;
```

  - 测试结果

result	result2	result3
3	3	-2

 版权所有 © 华为技术有限公司

CONVERT_TZ

- **功能描述**

  参考语法说明，本函数将日期时间string1（具有默认ISO时间戳格式'yyyy-MM-dd HH:mm:ss'）从时区string2转换为时区string3的值，结果以STRING类型返回。

- **语法说明**

  ```
  STRING CONVERT_TZ(string1, string2, string3)
  ```

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串，不符合格式的字符串会返回NULL。
string2	STRING	转换前时区。时区的格式应该是缩写如“PST”，全名如“Country A/City A”，或自定义ID如“GMT-08:00”。
string3	STRING	转换后时区。时区的格式应该是缩写如“PST”，全名如“Country A/City A”，或自定义ID如“GMT-08:00”。

- **示例**

  - 测试语句

    ```
    SELECT
       CONVERT_TZ(1970-01-01 00:00:00, UTC, Country A/City A) AS `result`,
          CONVERT_TZ(1997-04-25 10:00:00, UTC, GMT-08:00) AS `result2`
    FROM    testtable;
    ```

  - 测试结果

result	result2
1969-12-31 16:00:00	1997-04-25 02:00:00

FROM_UNIXTIME

- **功能描述**

  参考语法说明，本函数根据时间戳numeric和当前时区返回string格式的时间。

- **语法说明**

  ```
  STRING FROM_UNIXTIME(numeric[, string])
  ```

- **入参说明**

参数名	数据类型	参数说明
numeric	BIGINT	内部时间戳值，表示自'1970-01-01 00:00:00' UTC 以来的秒数，值可以由 UNIX_TIMESTAMP() 函数生成。
string	STRING	时间字符串格式。如果该参数不指定，则默认为'yyyy-MM-dd HH:mm:ss'。

 版权所有 © 华为技术有限公司

- **示例**
  - 测试语句
    ```
    SELECT
        FROM_UNIXTIME(44) AS `result`,
            FROM_UNIXTIME(44, 'yyyy:MM:dd') AS `result2`
    FROM    testtable;
    ```
  - 测试结果

result	result2
1970-01-01 08:00:44	1970:01:01

UNIX_TIMESTAMP

- **功能描述**

  以秒为单位获取当前的Unix时间戳。以BIGINT类型返回。
- **语法说明**
  ```
  BIGINT UNIX_TIMESTAMP()
  ```
- **入参说明**

  无。
- **示例**
  - 测试语句
    ```
    SELECT
        UNIX_TIMESTAMP() AS `result`
    FROM
        table;
    ```
  - 测试结果

result
1635401982

UNIX_TIMESTAMP(string1[, string2])

- **功能描述**

  参数语法说明，本函数将以string2格式的时间字符串string1转为Unix 时间戳（以秒为单位）。以BIGINT类型返回。
- **语法说明**
  ```
  BIGINT UNIX_TIMESTAMP(string1[, string2])
  ```
- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合string2参数格式的字符串语法会报错。
string2	STRING	时间字符串格式。如果不指定该参数，则默认为'yyyy-MM-dd HH:mm:ss' 。

 版权所有 © 华为技术有限公司

- **示例**
  - 测试语句

```
SELECT
    UNIX_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
        UNIX_TIMESTAMP('1997-04-25 00:00:10', 'yyyy-MM-dd HH:mm:ss') AS `result2`,
        UNIX_TIMESTAMP('1997-04-25 00:00:00') AS `result3`
FROM
    testtable;
```

  - 测试结果

result	result2	result3
861897600	861897610	861897600

TO_DATE

- **功能描述**

  参数语法说明，本函数将string2格式的日期字符串string1转换为DATE类型。

- **语法说明**

```
DATE TO_DATE(string1[, string2])
```

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合格式的字符串会执行报错。
string2	STRING	字符串格式。如果不指定该参数，则默认为'yyyy-MM-dd' 。

- **示例**
  - 测试语句

```
SELECT
    TO_DATE('1997-04-25') AS `result`,
        TO_DATE('1997:04:25', 'yyyy-MM-dd') AS `result2`,
        TO_DATE('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
    testtable;
```

  - 测试结果

result	result2	result3
1997-04-25	1997-04-25	1997-04-25

TO_TIMESTAMP

- **功能描述**

  将string2格式的日期时间字符串string1转换为TIMESTAMP类型返回。

- **语法说明**

```
TIMESTAMP TO_TIMESTAMP(string1[, string2])
```

 版权所有 © 华为技术有限公司

- **入参说明**

参数名	数据类型	参数说明
string1	STRING	SQL时间戳形式的字符串。不符合格式的字符串会返回NULL。
string2	STRING	日期字符串格式。如果该参数不指定，则默认为'yyyy-MM-dd HH:mm:ss' 。

- **示例**
  - 测试语句

```
SELECT
    TO_TIMESTAMP('1997-04-25', 'yyyy-MM-dd') AS `result`,
        TO_TIMESTAMP('1997-04-25 00:00:00') AS `result2`,
        TO_TIMESTAMP('1997-04-25 00:00:00', 'yyyy-MM-dd HH:mm:ss') AS `result3`
FROM
    testtable;
```

  - 测试结果

result	result2	result3
1997-04-25 00:00	1997-04-25 00:00	1997-04-25 00:00

4.5.2.4 条件函数

函数说明

表 4-37 条件函数

条件函数	函数说明
CASE value WHEN value1_1 [, value1_2]* THEN result1 [WHEN value2_1 [, value2_2]* THEN result2]* [ELSE resultZ] END	当value被包含在valueX_1、valueX_2.....中时，则返回结果resultX 仅返回匹配到的第一条结果 若都不匹配，如果提供了默认值resultZ，则返回resultZ，否则返回null
CASE WHEN condition1 THEN result1 [WHEN condition2 THEN result2]* [ELSE resultZ] END	当条件表达式conditionX为TRUE时，则返回resultX 仅返回匹配到的第一条结果 若都不为TRUE，如果提供了默认值resultZ，则返回resultZ，否则返回null

 版权所有 © 华为技术有限公司

条件函数	函数说明
NULLIF(value1, value2)	若两个值相同则返回null，否则返回value1 例如：NULLIF(5, 5)返回NULL NULLIF(5, 0)返回5
COALESCE(value1, value2 [, value3]*)	返回从左到右第一个不为null的参数的值 例如：COALESCE(NULL, 5)返回5
IF(condition, true_value, false_value)	若condition为TRUE则返回true_value，否则返回false_value 例如：IF(5 > 3, 5, 3)返回5
IS_ALPHA(string)	若string中的所有字符都是字母，则返回TRUE，否则返回FALSE
IS_DECIMAL(string)	若字符串可以转换为数值，则返回TRUE
IS_DIGIT(string)	若字符串中的所有字符都是数字，则返回TRUE。否则返回FALSE

4.5.2.5 类型转换函数

语法格式

```
CAST(value AS type)
```

语法说明

类型强制转换。

注意事项

若输入为NULL，则返回NULL。

示例

将amount值转换成整型。

```
insert into temp select cast(amount as INT) from source_stream;
```


版权所有 © 华为技术有限公司

表 4-38 类型转换函数示例

示例	说明	示例
cast(v1 as string)	将v1转换为字符串类型，v1可以是数值类型，TIMESTAMP/DATE/TIME。	表T1: | content (INT)

5	 语句： SELECT cast(content as varchar) FROM T1; 结果： "5"
cast (v1 as int)	将v1转换为int, v1可以是数值类型或字符类。	表T1: | content (STRING)

"5"	 语句： SELECT cast(content as int) FROM T1; 结果： 5
cast(v1 as timestamp)	将v1转换为timestamp类型，v1可以是字符串或DATE/TIME。	表T1: | content (STRING)

"2018-01-01 00:00:01"	 语句： SELECT cast(content as timestamp) FROM T1; 结果： 1514736001000
cast(v1 as date)	将v1转换为date类型， v1可以是字符串或者TIMESTAMP。	表T1: | content (TIMESTAMP)

1514736001000	 语句： SELECT cast(content as date) FROM T1; 结果： "2018-01-01"

说明

Flink作业不支持使用CAST将“BIGINT”转换为“TIMESTAMP”，可以使用to_timestamp进行转换。

 版权所有 © 华为技术有限公司

详细样例代码

```
/** source **/
CREATE
TABLE car_infos (cast_int_to_string int, cast_String_to_int string,
case_string_to_timestamp string, case_timestamp_to_date timestamp(3)) WITH (
  'connector.type' = 'dis',
  'connector.region' = 'xxxxx',
  'connector.channel' = 'dis-input',
  'format.type' = 'json'
);
/** sink **/
CREATE
TABLE cars_infos_out (cast_int_to_string string, cast_String_to_int
int, case_string_to_timestamp timestamp(3), case_timestamp_to_date date) WITH (
  'connector.type' = 'dis',
  'connector.region' = 'xxxxx',
  'connector.channel' = 'dis-output',
  'format.type' = 'json'
);
/** 统计car的静态信息 **/
INSERT
INTO
  cars_infos_out
SELECT
  cast(cast_int_to_string as string),
  cast(cast_String_to_int as int),
  cast(case_string_to_timestamp as timestamp),
  cast(case_timestamp_to_date as date)
FROM
  car_infos;
```

4.5.2.6 集合函数

函数说明

表 4-39 集合函数说明

集合函数	函数说明
CARDINALITY(array)	返回数组中元素个数
array ‘[’ integer ‘]’	返回数组索引为integer的元素。索引从1开始
ELEMENT(array)	返回数组中的唯一元素。 若数组为空，则返回null 若数组中元素个数大于1，则抛出异常
CARDINALITY(map)	返回map中键值对的条数
map ‘[’ key ‘]’	返回map中key所对应的值

 版权所有 © 华为技术有限公司

4.5.2.7 值构建函数

函数说明

表 4-40 值构建函数说明

值构建函数	函数说明
ROW(value1, [, value2]*) (value1, [, value2]*)	根据一系列值创建ROW
ARRAY ‘[’ value1 [, value2]* ‘]’	根据一系列值创建数组
MAP ‘[’ key1, value1 [, key2, value2]* ‘]’	根据一系列值创建MAP 其键值对为(key1, value1),(key2, value2)

4.5.2.8 属性访问函数

函数说明

表 4-41 属性访问函数说明

值接入函数	函数说明
tableName.compositeType.field	选择单个字段，通过名称访问Apache Flink复合类型（如Tuple，POJO等）的字段并返回其值。
tableName.compositeType.*	选择所有字段，将Apache Flink复合类型（如Tuple，POJO等）和其所有直接子类型转换为简单表示，其中每个子类型都是单独的字段。

4.5.2.9 Hash 函数

函数说明

表 4-42 Hash 函数说明

Hash函数	函数说明
MD5(string)	返回以32个十六进制数所表示的字符串的MD5哈希值 若字符串是null，则返回null
SHA1(string)	返回以40个十六进制所表示的字符串的SHA-1哈希值 若字符串是null，则返回null

版权所有 © 华为技术有限公司

Hash函数	函数说明
SHA224(string)	返回以56个十六进制数所表示的字符串的SHA-224哈希值 若字符串是null，则返回null
SHA256(string)	返回以64个十六进制数所表示的字符串的SHA-256哈希值 若字符串是null，则返回null
SHA384(string)	返回以96个十六进制数所表示的字符串的SHA-384哈希值 若字符串是null，则返回null
SHA512(string)	返回以128个十六进制数所表示的字符串的SHA-512哈希值 若字符串是null，则返回null
SHA2(string, hashLength)	返回使用SHA-2哈希函数族（SHA-224, SHA-256, SHA-384, or SHA-512）得到的哈希值 第一个参数string表示被哈希的字符串，第二个参数hashLength表示哈希值的长度（224、256、384、512） 若任意参数为null，则返回null

4.5.2.10 聚合函数

聚合函数是从一组输入值计算一个结果。例如使用COUNT函数计算SQL查询语句返回的记录行数。聚合函数如表4-43所示。

表 4-43 聚合函数表

函数	返回值类型	描述
COUNT([ALL] expression | DISTINCT expression1 [, expression2]*)	BIGINT	返回表达式不为NULL的输入行数。对每个值的一个唯一实例使用DISTINCT。
COUNT(*) COUNT(1)	BIGINT	返回元组个数
AVG([ALL | DISTINCT] expression)	DOUBLE	返回所有值的平均值。 对每个值的一个唯一实例使用DISTINCT。
SUM([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值的数值之和 对每个值的一个唯一实例使用DISTINCT
MAX([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值的最大值
MIN([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值的最小值

版权所有 © 华为技术有限公司

函数	返回值类型	描述
STDDEV_POP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的总体标准偏差
STDDEV_SAMP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的样本标准偏差
VAR_POP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的总体方差
VAR_SAMP([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的样本方差
COLLECT([ALL | DISTINCT] expression)	MULTISET	返回所有输入值的MULTISET
VARIANCE([ALL | DISTINCT] expression)	DOUBLE	返回所有输入值之间的数字字段的样本方差
FIRST_VALUE(expression)	数据实际类型	返回有序数据中的第一个数据
LAST_VALUE(expression)	数据实际类型	返回有序数据中的最后一个数据

4.5.2.11 表值函数

4.5.2.11.1 split_cursor

split_cursor表值函数可以将一行转多行，一列转为多列，仅支持在JOIN LATERAL TABLE中使用。

表 4-44 split_cursor 表值函数表

函数	返回值类型	描述
split_cursor(value, delimiter)	cursor	将字符串value按delimiter分隔为多行字符串。

示例

输入一条记录("student1", "student2, student3")，输出两条记录("student1", "student2") 和 ("student1", "student3") 。

```
create table s1(attr1 string, attr2 string) with (.....);
insert into s2 select  attr1, b1 from s1 left join lateral table(split_cursor(attr2, ',')) as T(b1) on true;
```

4.5.2.11.2 string_split

string_split函数，根据指定的分隔符将目标字符串拆分为子字符串，并返回子字符串列表。

 版权所有 © 华为技术有限公司

语法说明

```
string_split(target, separator)
```

表 4-45 string_split 参数说明

参数	数据类型	说明
target	STRING	待处理的目标字符串。 **说明** • 如果target为NULL，则返回一个空行。 • 如果target包含两个或多个连续出现的分隔符时，则返回长度为零的空子字符串。 • 如果target未包含指定分隔符，则返回目标字符串。
separator	VARCHAR	指定的分隔符，当前仅支持单字符分隔。

示例

1. 准备测试输入数据

   表 4-46 测试源表 disSource 数据和分隔符

target（STRING）	separator （VARCHAR）
test-flink	-
flink	-
one-two-ww-three	-

2. 输入测试SQL语句

```
create table disSource(
  target STRING,
  separator  VARCHAR
) with (
  "connector.type" = "dis",
  "connector.region" = "xxx",
  "connector.channel" = "ygj-dis-in",
  "format.type" = 'csv'
);

create table disSink(
  target STRING,
  item STRING
) with (
  'connector.type' = 'dis',
  'connector.region' = 'xxx',
  'connector.channel' = 'ygj-dis-out',
  'format.type' = 'csv'
);
```


版权所有 © 华为技术有限公司

```
insert into
  disSink
select
  target,
  item
from
  disSource,
lateral table(string_split(target, separator)) as T(item);
```

3. 查看测试结果

表 4-47 disSink 结果表数据

target（STRING）	item（STRING）
test-flink	test
test-flink	flink
flink	flink
one-two-ww-three	one
one-two-ww-three	two
one-two-ww-three	ww
one-two-ww-three	three

 版权所有 © 华为技术有限公司