数据湖探索

SQL 语法参考

文档版本 01

发布日期 2025-01-21

华为技术有限公司

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

目 录

1 Spark SQL 语法参考.....1
1.1 批作业 SQL 常用配置项说明.....1
1.2 批作业 SQL 语法概览.....3
1.3 Spark 开源命令支持说明.....5
1.4 数据库.....7
1.4.1 创建数据库.....7
1.4.2 删除数据库.....8
1.4.3 查看指定数据库.....9
1.4.4 查看所有数据库.....10
1.5 创建 OBS 表.....10
1.5.1 使用 DataSource 语法创建 OBS 表.....10
1.5.2 使用 Hive 语法创建 OBS 表.....17
1.6 创建 DLI 表.....23
1.6.1 使用 DataSource 语法创建 DLI 表.....23
1.6.2 使用 Hive 语法创建 DLI 表.....26
1.7 删除表.....31
1.8 查看表.....32
1.8.1 查看所有表.....32
1.8.2 查看建表语句.....32
1.8.3 查看表属性.....33
1.8.4 查看指定表所有列.....34
1.8.5 查看指定表所有分区.....35
1.8.6 查看表统计信息.....36
1.9 修改表.....36
1.9.1 添加列.....36
1.9.2 修改列注释.....37
1.10 分区表相关.....38
1.10.1 添加分区（只支持 OBS 表）.....38
1.10.2 重命名分区（只支持 OBS 表）.....40
1.10.3 删除分区.....41
1.10.4 指定筛选条件删除分区（只支持 OBS 表）.....44
1.10.5 修改表分区位置（只支持 OBS 表）.....48
1.10.6 更新表分区信息（只支持 OBS 表）.....49

1.10.7 REFRESH TABLE 刷新表元数据 50
1.11 导入数据 51
1.12 插入数据 55
1.13 清空数据 57
1.14 导出查询结果 58
1.15 表生命周期管理 59
1.15.1 创建表时指定表的生命周期 59
1.15.2 修改表生命周期的时间 61
1.15.3 禁止或恢复表的生命周期 62
1.16 跨源连接 HBase 表 64
1.16.1 创建表关联 HBase 64
1.16.2 插入数据至 HBase 表 66
1.16.3 查询 HBase 表 67
1.17 跨源连接 OpenTSDB 表 69
1.17.1 创建表关联 OpenTSDB 69
1.17.2 插入数据至 OpenTSDB 表 70
1.17.3 查询 OpenTSDB 表 71
1.18 跨源连接 DWS 表 72
1.18.1 创建表关联 DWS 72
1.18.2 插入数据至 DWS 表 74
1.18.3 查询 DWS 表 75
1.19 跨源连接 RDS 表 75
1.19.1 创建表关联 RDS 75
1.19.2 插入数据至 RDS 表 78
1.19.3 查询 RDS 表 79
1.20 跨源连接 CSS 表 80
1.20.1 创建表关联 CSS 80
1.20.2 插入数据至 CSS 表 81
1.20.3 查询 CSS 表 83
1.21 跨源连接 DCS 表 83
1.21.1 创建表关联 DCS 83
1.21.2 插入数据至 DCS 表 85
1.21.3 查询 DCS 表 87
1.22 跨源连接 DDS 表 87
1.22.1 创建表关联 DDS 87
1.22.2 插入数据至 DDS 表 89
1.22.3 查询 DDS 表 90
1.23 跨源连接 Oracle 表 90
1.23.1 创建表关联 Oracle 90
1.23.2 插入数据至 Oracle 表 92
1.23.3 查询 Oracle 表 93
1.24 视图 93

1.24.1 创建视图 93
1.24.2 删除视图 94
1.25 查看计划 94
1.26 数据权限管理 95
1.26.1 数据权限列表 95
1.26.2 创建角色 97
1.26.3 删除角色 98
1.26.4 绑定角色 98
1.26.5 解绑角色 99
1.26.6 显示角色 99
1.26.7 分配权限 100
1.26.8 回收权限 101
1.26.9 显示已授权限 102
1.26.10 显示所有角色和用户的绑定关系 102
1.27 数据类型 103
1.27.1 概述 103
1.27.2 原生数据类型 103
1.27.3 复杂数据类型 106
1.28 自定义函数 108
1.28.1 创建函数 108
1.28.2 删除函数 109
1.28.3 显示函数详情 110
1.28.4 显示所有函数 110
1.29 内置函数 111
1.29.1 日期函数 111
1.29.1.1 日期函数概览 111
1.29.1.2 add_months 114
1.29.1.3 current_date 115
1.29.1.4 current_timestamp 115
1.29.1.5 date_add 116
1.29.1.6 dateadd 117
1.29.1.7 date_sub 119
1.29.1.8 date_format 120
1.29.1.9 datediff 121
1.29.1.10 datediff1 122
1.29.1.11 datepart 124
1.29.1.12 datetrunc 125
1.29.1.13 day/dayofmonth 126
1.29.1.14 from_unixtime 127
1.29.1.15 from_utc_timestamp 128
1.29.1.16 getdate 129
1.29.1.17 hour 129

1.29.1.18 isdate 130
1.29.1.19 last_day 131
1.29.1.20 lastday 132
1.29.1.21 minute 133
1.29.1.22 month 134
1.29.1.23 months_between 135
1.29.1.24 next_day 136
1.29.1.25 quarter 138
1.29.1.26 second 138
1.29.1.27 to_char 139
1.29.1.28 to_date 141
1.29.1.29 to_date1 141
1.29.1.30 to_utc_timestamp 143
1.29.1.31 trunc 144
1.29.1.32 unix_timestamp 145
1.29.1.33 weekday 146
1.29.1.34 weekofyear 147
1.29.1.35 year 148
1.29.2 字符串函数 149
1.29.2.1 字符串函数概览 149
1.29.2.2 ascii 152
1.29.2.3 concat 153
1.29.2.4 concat_ws 154
1.29.2.5 char_matchcount 155
1.29.2.6 encode 156
1.29.2.7 find_in_set 157
1.29.2.8 get_json_object 158
1.29.2.9 instr 159
1.29.2.10 instr1 160
1.29.2.11 initcap 162
1.29.2.12 keyvalue 162
1.29.2.13 length 163
1.29.2.14 lengthb 164
1.29.2.15 levenshtein 165
1.29.2.16 locate 165
1.29.2.17 lower/lcase 166
1.29.2.18 lpad 167
1.29.2.19 ltrim 168
1.29.2.20 parse_url 169
1.29.2.21 printf 171
1.29.2.22 regexp_count 171
1.29.2.23 regexp_extract 172

1.29.2.24 replace 173
1.29.2.25 regexp_replace 174
1.29.2.26 regexp_replace1 176
1.29.2.27 regexp_instr 177
1.29.2.28 regexp_substr 178
1.29.2.29 repeat 179
1.29.2.30 reverse 180
1.29.2.31 rpad 180
1.29.2.32 rtrim 181
1.29.2.33 soundex 182
1.29.2.34 space 183
1.29.2.35 substr/substring 184
1.29.2.36 substring_index 185
1.29.2.37 split_part 185
1.29.2.38 translate 187
1.29.2.39 trim 187
1.29.2.40 upper/ucase 189
1.29.3 数学函数 189
1.29.3.1 数学函数概览 189
1.29.3.2 abs 192
1.29.3.3 acos 193
1.29.3.4 asin 194
1.29.3.5 atan 195
1.29.3.6 bin 195
1.29.3.7 bround 196
1.29.3.8 cbrt 197
1.29.3.9 ceil 198
1.29.3.10 conv 199
1.29.3.11 cos 200
1.29.3.12 cot1 201
1.29.3.13 degrees 202
1.29.3.14 e 202
1.29.3.15 exp 203
1.29.3.16 factorial 203
1.29.3.17 floor 204
1.29.3.18 greatest 205
1.29.3.19 hex 206
1.29.3.20 least 207
1.29.3.21 ln 208
1.29.3.22 log 209
1.29.3.23 log10 210
1.29.3.24 log2 210

1.29.3.25 median....211
1.29.3.26 negative....212
1.29.3.27 percentlie....213
1.29.3.28 percentlie_approx....214
1.29.3.29 pi....215
1.29.3.30 pmod....215
1.29.3.31 positive....216
1.29.3.32 pow....217
1.29.3.33 radians....218
1.29.3.34 rand....219
1.29.3.35 round....219
1.29.3.36 shiftleft....220
1.29.3.37 shiftright....221
1.29.3.38 shiftrightunsigned....222
1.29.3.39 sign....223
1.29.3.40 sin....224
1.29.3.41 sqrt....225
1.29.3.42 tan....226
1.29.4 聚合函数....227
1.29.4.1 聚合函数概览....227
1.29.4.2 avg....227
1.29.4.3 corr....228
1.29.4.4 count....229
1.29.4.5 covar_pop....230
1.29.4.6 covar_samp....231
1.29.4.7 max....232
1.29.4.8 min....233
1.29.4.9 percentile....233
1.29.4.10 percentile_approx....234
1.29.4.11 stddev_pop....235
1.29.4.12 stddev_samp....236
1.29.4.13 sum....237
1.29.4.14 variance/var_pop....238
1.29.4.15 var_samp....239
1.29.5 分析窗口函数....239
1.29.5.1 分析窗口函数概览....239
1.29.5.2 cume_dist....240
1.29.5.3 first_value....242
1.29.5.4 last_value....243
1.29.5.5 lag....245
1.29.5.6 lead....247
1.29.5.7 percent_rank....249

1.29.5.8 rank....250
1.29.5.9 row_number....251
1.29.6 其他函数....253
1.29.6.1 函数概览....253
1.29.6.2 decode1....254
1.29.6.3 javahash....255
1.29.6.4 max_pt....256
1.29.6.5 ordinal....256
1.29.6.6 trans_array....257
1.29.6.7 trunc_numeric....259
1.29.6.8 url_decode....260
1.29.6.9 url_encode....260
1.30 SELECT 基本语句....261
1.31 过滤 SELECT....263
1.31.1 WHERE 过滤子句....263
1.31.2 HAVING 过滤子句....263
1.32 排序 SELECT....264
1.32.1 ORDER BY....264
1.32.2 SORT BY....264
1.32.3 CLUSTER BY....265
1.32.4 DISTRIBUTE BY....265
1.33 分组 SELECT....266
1.33.1 按列 GROUP BY....266
1.33.2 用表达式 GROUP BY....267
1.33.3 GROUP BY 中使用 HAVING 过滤....267
1.33.4 ROLLUP....268
1.33.5 GROUPING SETS....269
1.34 连接操作 SELECT....270
1.34.1 内连接....270
1.34.2 左外连接....270
1.34.3 右外连接....271
1.34.4 全外连接....271
1.34.5 隐式连接....272
1.34.6 笛卡尔连接....272
1.34.7 左半连接....273
1.34.8 不等值连接....273
1.35 子查询....274
1.35.1 WHERE 嵌套子查询....274
1.35.2 FROM 子句嵌套子查询....275
1.35.3 HAVING 子句嵌套子查询....275
1.35.4 多层嵌套子查询....276
1.36 别名 SELECT....276

1.36.1 表别名 276
1.36.2 列别名 277
1.37 集合运算 SELECT 277
1.37.1 UNION 278
1.37.2 INTERSECT 278
1.37.3 EXCEPT 279
1.38 WITH...AS 279
1.39 CASE...WHEN 280
1.39.1 简单 CASE 函数 280
1.39.2 CASE 搜索函数 280
1.40 OVER 子句 281
2 Flink Opensource SQL1.12 语法参考 283
2.1 SQL 语法约束与定义 283
2.1.1 语法支持类型 283
2.1.2 语法定义 283
2.1.2.1 DDL 语法定义 283
2.1.2.1.1 CREATE TABLE 语句 283
2.1.2.1.2 CREATE VIEW 语句 286
2.1.2.1.3 CREATE FUNCTION 语句 286
2.1.2.2 DML 语法定义 287
2.2 Flink OpenSource SQL1.12 语法概览 289
2.3 数据定义语句 DDL 290
2.3.1 创建源表 290
2.3.1.1 DataGen 源表 290
2.3.1.2 DWS 源表 293
2.3.1.3 Hbase 源表 297
2.3.1.4 JDBC 源表 301
2.3.1.5 Kafka 源表 306
2.3.1.6 MySQL CDC 源表 317
2.3.1.7 Postgres CDC 源表 321
2.3.1.8 Redis 源表 325
2.3.1.9 Upsert Kafka 源表 331
2.3.2 创建结果表 334
2.3.2.1 BlackHole 结果表 335
2.3.2.2 ClickHouse 结果表 336
2.3.2.3 DWS 结果表 339
2.3.2.4 Elasticsearch 结果表 344
2.3.2.5 Hbase 结果表 350
2.3.2.6 JDBC 结果表 355
2.3.2.7 Kafka 结果表 360
2.3.2.8 Print 结果表 368
2.3.2.9 Redis 结果表 370

2.3.2.10 Upsert Kafka 结果表 380
2.3.2.11 FileSystem 结果表 383
2.3.3 创建维表 387
2.3.3.1 DWS 维表 387
2.3.3.2 Hbase 维表 392
2.3.3.3 JDBC 维表 397
2.3.3.4 Redis 维表 402
2.3.4 Format 407
2.3.4.1 Avro Format 407
2.3.4.2 Canal Format 410
2.3.4.3 Confluent Avro Format 413
2.3.4.4 CSV Format 415
2.3.4.5 Debezium Format 417
2.3.4.6 JSON Format 419
2.3.4.7 Maxwell Format 423
2.3.4.8 Raw Format 425
2.4 数据操作语句 DML 427
2.4.1 SELECT 427
2.4.2 集合操作 430
2.4.3 窗口 431
2.4.4 JOIN 439
2.4.5 OrderBy & Limit 441
2.4.6 Top-N 442
2.4.7 去重 443
2.5 函数 443
2.5.1 自定义函数 444
2.5.2 内置函数 447
2.5.2.1 数学运算函数 447
2.5.2.2 字符串函数 453
2.5.2.3 时间函数 458
2.5.2.4 条件函数 480
2.5.2.5 类型转换函数 481
2.5.2.6 集合函数 483
2.5.2.7 值构建函数 484
2.5.2.8 属性访问函数 484
2.5.2.9 Hash 函数 485
2.5.2.10 聚合函数 485
2.5.2.11 表值函数 486
2.5.2.11.1 string_split 486
3 Flink Opensource SQL1.10 语法参考 489
3.1 SQL 语法约束与定义 489
3.1.1 语法支持类型 489

3.1.2 语法定义 489
3.1.2.1 DDL 语法定义 489
3.1.2.1.1 CREATE TABLE 语句 489
3.1.2.1.2 CREATE VIEW 语句 492
3.1.2.1.3 CREATE FUNCTION 语句 492
3.1.2.2 DML 语法定义 493
3.2 Flink OpenSource SQL1.10 语法概览 495
3.3 数据定义语句 DDL 496
3.3.1 创建源表 496
3.3.1.1 Kafka 源表 496
3.3.1.2 DIS 源表 499
3.3.1.3 JDBC 源表 501
3.3.1.4 DWS 源表 503
3.3.1.5 Redis 源表 505
3.3.1.6 Hbase 源表 506
3.3.1.7 userDefined 源表 508
3.3.2 创建结果表 509
3.3.2.1 ClickHouse 结果表 509
3.3.2.2 Kafka 结果表 512
3.3.2.3 Upsert Kafka 结果表 513
3.3.2.4 DIS 结果表 515
3.3.2.5 JDBC 结果表 516
3.3.2.6 DWS 结果表 518
3.3.2.7 Redis 结果表 520
3.3.2.8 SMN 结果表 523
3.3.2.9 Hbase 结果表 525
3.3.2.10 Elasticsearch 结果表 526
3.3.2.11 OpenTSDB 结果表 529
3.3.2.12 userDefined 结果表 531
3.3.2.13 Print 结果表 532
3.3.2.14 FileSytem 结果表 534
3.3.3 创建维表 536
3.3.3.1 创建 JDBC 维表 536
3.3.3.2 创建 DWS 维表 539
3.3.3.3 创建 Hbase 维表 541
3.4 数据操作语句 DML 543
3.4.1 SELECT 543
3.4.2 集合操作 546
3.4.3 窗口 547
3.4.4 JOIN 552
3.4.5 OrderBy & Limit 554
3.4.6 Top-N 555

3.4.7 去重 556
3.5 函数 557
3.5.1 自定义函数 557
3.5.2 内置函数 561
3.5.2.1 数学运算函数 561
3.5.2.2 字符串函数 567
3.5.2.3 时间函数 573
3.5.2.4 条件函数 595
3.5.2.5 类型转换函数 595
3.5.2.6 集合函数 598
3.5.2.7 值构建函数 598
3.5.2.8 属性访问函数 598
3.5.2.9 Hash 函数 599
3.5.2.10 聚合函数 599
3.5.2.11 表值函数 600
3.5.2.11.1 split_cursor 600
3.5.2.11.2 string_split 601
4 历史版本 603
4.1 Flink SQL 语法参考 603
4.1.1 SQL 语法约束与定义 603
4.1.2 流作业 SQL 语法概览 604
4.1.3 创建输入流 605
4.1.3.1 DIS 输入流 605
4.1.3.2 DMS 输入流 610
4.1.3.3 MRS Kafka 输入流 610
4.1.3.4 开源 Kafka 输入流 613
4.1.3.5 OBS 输入流 616
4.1.4 创建输出流 618
4.1.4.1 MRS OpenTSDB 输出流 618
4.1.4.2 CSS Elasticsearch 输出流 619
4.1.4.3 DCS 输出流 622
4.1.4.4 DDS 输出流 623
4.1.4.5 DIS 输出流 625
4.1.4.6 DMS 输出流 627
4.1.4.7 DWS 输出流（通过 JDBC 方式） 627
4.1.4.8 DWS 输出流（通过 OBS 转储方式） 629
4.1.4.9 MRS HBase 输出流 632
4.1.4.10 MRS Kafka 输出流 634
4.1.4.11 开源 Kafka 输出流 636
4.1.4.12 文件系统输出流(推荐） 637
4.1.4.13 OBS 输出流 640
4.1.4.14 RDS 输出流 643

4.1.4.15 SMN 输出流 645
4.1.5 创建中间流 647
4.1.6 创建维表 647
4.1.6.1 创建 Redis 表 647
4.1.6.2 创建 RDS 表 648
4.1.7 自拓展生态 651
4.1.7.1 自拓展输入流 651
4.1.7.2 自拓展输出流 652
4.1.8 数据类型 653
4.1.9 内置函数 657
4.1.9.1 数学运算函数 657
4.1.9.2 字符串函数 661
4.1.9.3 时间函数 674
4.1.9.4 类型转换函数 677
4.1.9.5 聚合函数 680
4.1.9.6 表值函数 684
4.1.9.7 其他函数 685
4.1.10 自定义函数 685
4.1.11 地理函数 689
4.1.12 SELECT 696
4.1.13 条件表达式 699
4.1.14 窗口 701
4.1.15 流表 JOIN 703
4.1.16 配置时间模型 704
4.1.17 CEP 模式匹配 706
4.1.18 StreamingML 710
4.1.18.1 异常检测 710
4.1.18.2 时间序列预测 711
4.1.18.3 实时聚类 713
4.1.18.4 深度学习模型预测 714
4.1.19 保留关键字 716
5 标示符 735
5.1 aggregate_func 735
5.2 alias 735
5.3 attr_expr 736
5.4 attr_expr_list 737
5.5 attrs_value_set_expr 737
5.6 boolean_expression 738
5.7 col 738
5.8 col_comment 738
5.9 col_name 738
5.10 col_name_list 739

5.11 condition 740
5.12 condition_list 742
5.13 cte_name 742
5.14 data_type 743
5.15 db_comment 743
5.16 db_name 743
5.17 else_result_expression 743
5.18 file_format 743
5.19 file_path 744
5.20 function_name 744
5.21 groupby_expression 744
5.22 having_condition 745
5.23 input_expression 746
5.24 join_condition 747
5.25 non_equi_join_condition 748
5.26 number 748
5.27 partition_col_name 748
5.28 partition_col_value 749
5.29 partition_specs 749
5.30 property_name 749
5.31 property_value 749
5.32 regex_expression 750
5.33 result_expression 750
5.34 select_statement 750
5.35 separator 750
5.36 sql_containing_cte_name 750
5.37 sub_query 751
5.38 table_comment 751
5.39 table_name 751
5.40 table_properties 751
5.41 table_reference 752
5.42 when_expression 752
5.43 where_condition 752
5.44 window_function 753
6 运算符 754
6.1 关系运算符 754
6.2 算术运算符 755
6.3 逻辑运算符 756

1 Spark SQL 语法参考

1.1 批作业 SQL 常用配置项说明

本章节为您介绍DLI 批作业SQL语法的常用配置项。

表 1-1 常用配置项

名称	默认值	描述
spark.sql.files.maxRecordsPerFile	0	要写入单个文件的最大记录数。如果该值为零或为负，则没有限制。
spark.sql.shuffle.partitions	200	为连接或聚合过滤数据时使用的默认分区数。
spark.sql.dynamicPartitionOverwrite.enabled	false	当前配置设置为“false”时，DLI在覆盖写之前，会删除所有符合条件的分区。例如，分区表中有一个“2021-01”的分区，当使用INSERT OVERWRITE语句向表中写入“2021-02”这个分区的数据时，会把“2021-01”的分区数据也覆盖掉。 当前配置设置为“true”时，DLI不会提前删除分区，而是在运行时覆盖那些有数据写入的分区。
spark.sql.files.maxPartitionBytes	134217728	读取文件时要打包到单个分区中的最大字节数。
spark.sql.badRecordsPath	-	Bad Records的路径。
dli.sql.sqlasync.enabled	true	DDL和DCL语句是否异步执行，值为“true”时启用异步执行。
dli.sql.job.timeout	-	设置作业运行超时时间，超时取消。单位：秒。

名称	默认值	描述
spark.sql.keep.distinct.expandThreshold	-	● **参数说明：** 对于包含count(distinct)的多维分析（with cube）的查询场景，spark典型的执行计划是将cube使用expand算子来实现，但该操作会导致查询膨胀，为了避免出现查询膨胀，建议执行如下配置： – spark.sql.keep.distinct.expandThreshold：默认值：-1，即使用Spark默认的expand算子。 设置具体数值：即代表定义了查询膨胀的阈值（例如512），超过该阈值count(distinct)使用distinct聚合算子来执行，不再使用expand算子。 – spark.sql.distinct.aggregator.enabled：强制使用distinct聚合算子的开关。配置为true时不再根据spark.sql.keep.distinct.expandThreshold来判断。 ● **适用场景：**包含count(distinct)的多维分析（with cube）的查询场景，可能包含多个count(distinct)，且包含cube/roll up ● **典型场景示例：** `SELECT a1, a2, count(distinct b), count(distinct c) FROM test_distinct group by a1, a2 with cube`
spark.sql.distinct.aggregator.enabled	false	
spark.sql.optimizer.dynamicPartitionPruning.enabled	true	该配置项用于启用或禁用动态分区修剪。在执行SQL查询时，动态分区修剪可以帮助减少需要扫描的数据量，提高查询性能。 ● 配置为true时，代表启用动态分区修剪，SQL会在查询中自动检测并删除那些不满足WHERE子句条件的分区，适用于在处理具有大量分区的表时。 ● 如果SQL查询中包含大量的嵌套left join操作，并且表有大量的动态分区时，这可能会导致在数据解析时消耗大量的内存资源，导致Driver节点的内存不足，并触发频繁的Full GC。 在这种情况下，可以配置该参数为false即禁用动态分区修剪优化，有助于减少内存使用，避免内存溢出和频繁的Full GC。 但禁用此优化可能会降低查询性能，禁用后Spark将不会自动修剪掉那些不满足条件的分区。

1.2 批作业 SQL 语法概览

本章节介绍了目前DLI所提供的Spark 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。

表 1-2 批作业 SQL 语法

语法分类	操作链接
数据库相关语法	创建数据库
	删除数据库
	查看指定数据库
	查看所有数据库
创建OBS表相关语法	使用DataSource语法创建OBS表
	使用Hive语法创建OBS表
删除表相关语法	删除表
查看表相关语法	查看所有表
	查看建表语句
	查看表属性
	查看指定表所有列
	查看指定表所有分区
	查看表统计信息
修改表相关语法	添加列
分区表相关语法	添加分区（只支持OBS表）
	重命名分区
	删除分区
	修改表分区位置（只支持OBS表）
	更新表分区信息（只支持OBS表）
导入数据相关语法	导入数据
插入数据相关语法	插入数据
清空数据相关语法	清空数据
导出查询结果相关语法	导出查询结果
跨源连接HBase表相关语法	创建表关联HBase
	插入数据至HBase表

语法分类	操作链接
	查询HBase表
跨源连接OpenTSDB表相关语法	创建表关联OpenTSDB
	插入数据至OpenTSDB
	查询OpenTSDB表
跨源连接DWS表相关语法	创建表关联DWS
	插入数据至DWS表
	查询DWS表
跨源连接RDS表相关语法	创建表关联RDS
	插入数据至RDS表
	查询RDS表
跨源连接CSS表相关语法	创建表关联CSS
	插入数据至CSS表
	查询CSS表
跨源连接DCS表相关语法	创建表关联DCS
	插入数据至DCS表
	查询DCS表
跨源连接DDS表相关语法	创建表关联DDS
	插入数据至DDS表
	查询DDS表
视图相关语法	创建视图
	删除视图
查看计划相关语法	查看计划
数据权限相关语法	创建角色
	删除角色
	绑定角色
	解绑角色
	显示角色
	分配权限
	回收权限
	显示已授权限

语法分类	操作链接
	显示所有角色和用户的绑定关系
自定义函数相关语法	创建函数
	删除函数
	显示函数详情
	显示所有函数

1.3 Spark 开源命令支持说明

本章节介绍了目前DLI对开源的Spark SQL语法的支持情况。详细的语法、参数说明，示例等信息请参考Spark官方文档。

表 1-3 DLI Spark 开源命令支持说明

功能描述	语法示例	DLI Spark 2.4.5	DLI Spark 3.3.1
创建数据库	CREATE DATABASE testDB;	支持	支持
创建DLI表	create table if not exists testDB.testTable1(id int, age int, money double);	支持	支持
创建OBS表	create table if not exists testDB.testTable2(id int, age int, money double) LOCATION 'obs://bucketName/filePath' partitioned by (dt string) row format delimited fields terminated by ',' STORED as TEXTFILE;	支持	支持
插入测试数据	insert into table testDB.testTable2 values (1, 18, 3.14, "20240101"), (2, 18, 3.15, "20240102");	支持	支持

功能描述	语法示例	DLI Spark 2.4.5	DLI Spark 3.3.1
修改数据库属性	ALTER DATABASE testDB SET DBPROPERTIES ('Edited-by' = 'John');	不支持	不支持
修改数据库在OBS上的文件存放路径	ALTER DATABASE testDB SET LOCATION 'obs://bucketName/filePath';	不支持	不支持
修改表名	ALTER TABLE testDB.testTable1 RENAME TO testDB.testTable101;	支持	支持
修改表的分区名	ALTER TABLE testDB.testTable2 PARTITION (dt='20240101') RENAME TO PARTITION (dt='20240103'); 只支持OBS表的分区名，且OBS上的文件存储路径不会变。	支持	支持
添加列	ALTER TABLE testDB.testTable1 ADD COLUMNS (name string);	支持	支持
删除列	ALTER TABLE testDB.testTable1 DROP columns (money);	不支持	不支持
修改列名称	ALTER TABLE testDB.testTable1 RENAME COLUMN age TO years_of_age;	不支持	不支持
修改列注释	ALTER TABLE testDB.testTable1 ALTER COLUMN age COMMENT "new comment";	不支持	支持
替换指定列	ALTER TABLE testDB.testTable1 REPLACE COLUMNS (name string, ID int COMMENT 'new comment');	不支持	不支持
自定义表属性	ALTER TABLE testDB.testTable1 SET TBLPROPERTIES ('aaa' = 'bbb');	支持	支持
添加或修改表注释	ALTER TABLE testDB.testTable1 SET TBLPROPERTIES ('comment' = 'test');	支持	不支持
修改表的存储格式	ALTER TABLE testDB.testTable1 SET fileformat csv;	不支持	不支持
删除表属性	ALTER TABLE testDB.testTable1 UNSET TBLPROPERTIES ('test');	支持	支持
删除表的注释	ALTER TABLE testDB.testTable1 UNSET TBLPROPERTIES ('comment');	支持	支持

功能描述	语法示例	DLI Spark 2.4.5	DLI Spark 3.3.1
展示列信息	DESCRIBE database_name.table_name col_name; 示例：DESCRIBE testDB.testTable1 id;	支持	支持
展示列信息	DESCRIBE table_name table_name.col_name; 示例：DESCRIBE testTable1 testTable1.id; 仅支持查看当前数据库下表的列信息。	支持	支持
返回查询语句的元数据信息	DESCRIBE QUERY SELECT age, sum(age) FROM testDB.testTable1 GROUP BY age;	不支持	支持
返回插入数据的元数据信息	DESCRIBE QUERY VALUES(100, 10, 10000.20D) AS testTable1(id, age, money);	不支持	支持
返回表的元数据信息	DESCRIBE QUERY TABLE testTable1;	不支持	支持
返回表中某列的元数据信息	DESCRIBE FROM testTable1 SELECT age;	不支持	支持
返回表的建表语句	SHOW CREATE TABLE testDB.testTable1 AS SERDE; Spark 3.3.1 执行该语句时，仅适用于查询Hive表的建表语句。	不支持	支持
返回表的建表语句	SHOW CREATE TABLE testDB.testTable1;	支持	支持

1.4 数据库

1.4.1 创建数据库

功能描述

创建数据库。

语法格式

```
CREATE [DATABASE | SCHEMA] [IF NOT EXISTS] db_name
  [COMMENT db_comment]
  [WITH DBPROPERTIES (property_name=property_value, ...)];
```

关键字

- IF NOT EXISTS：所需创建的数据库已存在时使用，可避免系统报错。
- COMMENT：对数据库的描述。
- DBPROPERTIES：数据库的属性，且属性名和属性值成对出现。

参数说明

表 1-4 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以数字和下划线开头。
db_comment	数据库描述。
property_name	数据库属性名。
property_value	数据库属性值。

注意事项

- DATABASE与SCHEMA两者没有区别，可替换使用，建议使用DATABASE。
- “default”为内置数据库，不能创建名为“default”的数据库。

示例

1. 队列是使用DLI服务的基础，执行SQL前需要先创建队列。
2. 在DLI管理控制台，单击左侧导航栏中的“SQL编辑器”，可进入SQL作业“SQL编辑器”页面。
3. 在“SQL编辑器”页面右侧的编辑窗口中，输入如下创建数据库的SQL语句，单击“执行”。阅读并同意隐私协议，单击“确定”。

   若testdb数据库不存在，则创建数据库testdb。

   ```
   CREATE DATABASE IF NOT EXISTS testdb;
   ```

1.4.2 删除数据库

功能描述

删除数据库。

语法格式

```
DROP [DATABASE | SCHEMA] [IF EXISTS] db_name [RESTRICT|CASCADE];
```

关键字

IF EXISTS：所需删除的数据库不存在时使用，可避免系统报错。

注意事项

- DATABASE与SCHEMA两者没有区别，可替换使用，建议使用DATABASE。
- RESTRICT表示如果该database不为空（有表存在），DROP操作会报错，执行失败，RESTRICT是默认逻辑。
- CASCADE表示即使该database不为空（有表存在），DROP也会级联删除下面的所有表，需要谨慎使用该功能。

参数说明

表 1-5 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。

示例

1. 已参考**示例**中描述创建对应的数据库，如testdb。
2. 若存在testdb数据库，则删除数据库testdb。

```
DROP DATABASE IF EXISTS testdb;
```

1.4.3 查看指定数据库

功能描述

查看指定数据库的相关信息，包括数据库名称、数据库的描述等。

语法格式

```
DESCRIBE DATABASE [EXTENDED] db_name;
```

关键字

EXTENDED：除了显示上述信息外，还会额外显示数据库的属性信息。

参数说明

表 1-6 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。

注意事项

如果所要查看的数据库不存在，则系统报错。

示例

1. 已参考**示例**中描述创建对应的数据库，如testdb。
2. 查看testdb数据库的相关信息。
   ```
   DESCRIBE DATABASE testdb;
   ```

1.4.4 查看所有数据库

功能描述

查看当前工程下所有的数据库。

语法格式

```
SHOW [DATABASES | SCHEMAS] [LIKE regex_expression];
```

关键字

无。

参数说明

表 1-7 参数说明

参数	描述
regex_expression	数据库名称。

注意事项

DATABASES与SCHEMAS是等效的，都将返回所有的数据库名称。

示例

查看当前的所有数据库。

```
SHOW DATABASES;
```

查看当前的所有以test开头的数据库。

```
SHOW DATABASES LIKE "test.*";
```

1.5 创建 OBS 表

1.5.1 使用 DataSource 语法创建 OBS 表

功能描述

本节介绍使用DataSource语法创建OBS表。

DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

说明

推荐使用OBS并行文件系统进行存储。并行文件系统是一种高性能文件系统，提供毫秒级别访问时延，TB/s级别带宽和百万级别的IOPS，适用于大数据交互式分析场景。

注意事项

- 创建表时不会统计大小。
- 添加数据时会修改大小至0。
- 如需查看表大小可以通过OBS查看。
- CTAS建表语句不能指定表的属性。
- **OBS目录下包含子目录的场景：**

  创建表时，若指定路径为OBS上的目录，且该目录下包含子目录（或嵌套子目录），则子目录下的所有文件类型及其内容也是表内容。

  您需要保证所指定的目录及其子目录下所有文件类型和建表语句中指定的存储格式一致，所有文件内容和表中的字段一致，否则查询将报错。

  您可以在建表语句OPTIONS中设置“multiLevelDirEnable”为true以查询子目录下的内容，此参数默认值为false（注意，此配置项为表属性，请谨慎配置。Hive表不支持此配置项)。
- **关于分区表的使用说明：**
  - 创建分区表时，PARTITIONED BY中指定分区列必须是表中的列，且必须在Column列表中指定类型。分区列只支持string, boolean, tinyint, smallint, short, int, bigint, long, decimal, float, double, date, timestamp类型。
  - 创建分区表时，分区字段必须是表字段的最后一个字段或几个字段，且多分区字段的顺序也必须对应。否则将出错。
  - 单表分区数最多允许200000个。
  - CTAS建表语句不支持创建分区表。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  USING file_format
  [OPTIONS (path 'obs_path', key1=val1, key2=val2, ...)]
  [PARTITIONED BY (col_name1, col_name2, ...)]
  [COMMENT table_comment]
  [AS select_statement]
```

关键字

- IF NOT EXISTS：指定该关键字以避免表已经存在时报错。
- USING：指定存储格式。
- OPTIONS：指定建表时的属性名与属性值。
- COMMENT：字段或表描述。
- PARTITIONED BY：指定分区字段。
- AS：使用CTAS创建表。

参数说明

表 1-8 参数说明

参数	是否必选	描述
db_name	否	Database名称。 由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以数字和下划线开头。
table_name	是	Database中的待创建的表名。 由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以数字和下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。 特殊字符需要使用单引号（ '' ）包围起来。 表名对大小写不敏感，即不区分大小写。
col_name	是	以逗号分隔的带数据类型的列名。 列名由字母、数字和下划线（ _ ）组成。不能是纯数字，且至少包含一个字母。 列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。
col_comment	否	列字段描述。仅支持字符串常量。
file_format	是	file_format是用于创建表的输入格式。支持orc，parquet，json，csv，avro类型。
path	是	数据文件所在的OBS存储路径，推荐使用OBS并行文件系统存储。 格式：obs://bucketName/tblPath bucketName即桶名称。 tblPath是目录名称。目录后不需要指定文件名。 更多建表时的属性名与属性值请参考**表1-9**。 file_format为csv时表的属性名与属性值请参考**表1-9**和**表1-10**。 当OBS的目录下文件夹与文件同名时，创建OBS表指向的路径会优先指向文件而非文件夹。
table_comment	否	表描述信息。仅支持字符串常量。
select_statement	否	用于CTAS命令，将源表的select查询结果或某条数据插入到新创建的OBS表中。

表 1-9 OPTIONS 参数描述

参数	是否必选	描述
path	否	指定的表路径，即OBS存储路径。
multiLevelDirEnable	否	嵌套子目录场景下，是否迭代查询子目录中的数据。当配置为true时，查询该表时会迭代读取该表路径中所有文件，包含子目录中的文件。 默认值：false
dataDelegated	否	是否需要在删除表或分区时，清除path路径下的数据。 默认值：false
compression	否	指定压缩格式。一般为parquet格式时指定该参数，推荐使用'zstd'压缩格式。

当file_format为csv时，支持设置以下OPTIONS参数。

表 1-10 CSV 数据格式 OPTIONS 参数说明

参数	是否必选	描述
delimiter	否	数据分隔符。 默认值：逗号（即",”）
quote	否	引用字符。 默认值：双引号（即“"”）
escape	否	转义字符。 默认值：反斜杠（即“\”）
multiLine	否	列数据中是否包含回车符或转行符，true为包含，false为不包含。 默认值：false
dateFormat	否	指定CSV文件中date字段的日期格式。 默认值：yyyy-MM-dd
timestampFormat	否	指定CSV文件中timestamp字段的日期格式。 默认值： yyyy-MM-dd HH:mm:ss

参数	是否必选	描述
mode	否	指定解析CSV时的模式，有三种模式。默认值：PERMISSIVE ● PERMISSIVE：宽容模式，遇到错误的字段时，设置该行整行为Null ● DROPMALFORMED: 遇到错误的字段时，丢弃整行。 ● FAILFAST：报错模式，遇到错误的字段时直接报错。
header	否	CSV是否包含表头信息，true表示包含表头信息，false为不包含。 默认值：false
nullValue	否	设置代表null的字符，例如，nullValue="nl"表示设置nl代表null。
comment	否	设置代表注释开头的字符，例如，comment='#'表示以#开头的行为注释。
compression	否	设置数据的压缩格式。目前支持gzip、bzip2、deflate压缩格式，若不希望压缩，则输入none。 默认值：none
encoding	否	数据的编码格式。支持utf-8，gb2312，gbk三种，如果不填写，则默认为utf-8。 默认值：utf-8

示例 1：创建 OBS 非分区表

示例说明：创建名为table1的OBS非分区表，使用USING关键字指定该表的存储格式为orc格式。

在您的实际使用中，可以将obs表存储为parquet、json、avro等类型。

```
CREATE TABLE IF NOT EXISTS table1 (
   col_1   STRING,
   col_2   INT)
USING orc
OPTIONS (path 'obs://bucketName/filePath');
```

示例 2：创建 OBS 分区表

示例说明：创建一个名为student的分区表，该分区表使用院系编号（facultyNo）和班级编号（classNo）进行分区。该student表会同时按照不同的院系编号（facultyNo）和不同的班级编号（classNo）分区。

在实际的使用过程中，您可以选择合适的分区字段并将其添加到PARTITIONED BY关键字后的括号内。

```
CREATE TABLE IF NOT EXISTS student (
   Name        STRING,
   facultyNo   INT,
```



```
  classNo     INT)
USING csv
OPTIONS (path 'obs://bucketName/filePath')
PARTITIONED BY (facultyNo, classNo);
```

示例 3：使用 CTAS 将源表的全部数据或部分数据创建新的 OBS 非分区表

示例说明：根据**示例1：创建OBS非分区表**中创建的OBS表table1，使用CTAS语法将table1中的数据复制到table1_ctas表中。

在使用CTAS建表的时候，可以忽略被复制的表在建表时所使用的语法，即不论在创建table1时使用的是何种语法，都可以使用DataSource语法的CTAS创建table1_ctas。

此外，本例中table1中OBS表的存储格式为orc，而table1_ctas表的存储格式可以为parquet，即CTAS创建的表存储格式可以不同于原表。

在AS关键字后使用SELECT语句选择需要的数据插入到table1_ctas表中。

SELECT语法为：SELECT <列名称> FROM <表名称> WHERE <相关筛选条件>。

- 示例中使用“SELECT * FROM table1”，'*'表示会从table1中选择所有列，并将table1中所有数据插入到table1_ctas表中。

  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  USING parquet
  OPTIONS (path 'obs:// bucketName/filePath')
  AS
  SELECT  *
  FROM    table1;
  ```

- 若需要按照自定义方式筛选数据插入table1_ctas中，可以使用如下的SELECT语句“SELECT col_1 FROM table1 WHERE col_1 = 'Ann' ”，这样就可以通过执行SELECT语句从table1中单独选定col_1，并只将其中值等于'Ann'的数据插入到table1_ctas中。

  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  USING parquet
  OPTIONS (path 'obs:// bucketName/filePath')
  AS
  SELECT  col_1
  FROM    table1
  WHERE   col_1 = 'Ann';
  ```

示例 4：创建 OBS 非分区表，并自定义列字段数据类型

示例说明：创建名为table2的OBS非分区表，您可以根据业务需求自定义列字段的原生数据类型：

- 与文字字符有关可以使用STRING、CHAR或者VARCHAR。
- 与时间有关的可以使用TIMESTAMP、DATE。
- 与整数有关的可以使用INT、SMALLINT/SHORT、BIGINT/LONG、TINYINT。
- 涉及小数运算可以使用FLOAT、DOUBLE、DECIMAL。
- 若数据只涉及逻辑开关可以使用BOOLEAN类型。

具体使用方法与明细可以参照“数据类型 > 原生数据类型”。

```
CREATE TABLE IF NOT EXISTS table2 (
  col_01  STRING,
  col_02  CHAR (2),
  col_03  VARCHAR (32),
  col_04  TIMESTAMP,
  col_05  DATE,
  col_06  INT,
```



```
    col_07  SMALLINT,
    col_08  BIGINT,
    col_09  TINYINT,
    col_10  FLOAT,
    col_11  DOUBLE,
    col_12  DECIMAL (10, 3),
    col_13  BOOLEAN
)
USING parquet
OPTIONS (path 'obs://bucketName/filePath');
```

示例 5：创建 OBS 分区表，自定义表的 OPTIONS 参数

示例说明：创建OBS表时支持自定义属性名与属性值，OPTIONS参数说明可参考**表1-9**。

本例创建名为table3并以col_2为分区依据的OBS分区表。在OPTIONS中配置path、multiLevelDirEnable、dataDelegated和compression。

- path：OBS存储路径，本例为“obs ://bucketName/filePath”，其中的bucketName为您存储时所使用桶名称，filePath为您实际使用的目录名称；
- 请注意大数据场景建议使用OBS并行文件系统进行存储；
- multiLevelDirEnable：本例设置为true，表示查询该表时会迭代读取表路径中的所有文件和子目录文件，若不需要此项配置可以设置为false或不设置（默认为false）；
- dataDelegated：本例设置为true，表示在删除表或相关分区时，会一并清除该path路径下的所有数据，若不需要此项配置可以设置为false或不设置（默认为false）；
- compression：当创建的OBS表需要压缩时，可以使用compression关键字来配置压缩格式，本例中就使用了zstd压缩格式。

```
CREATE TABLE IF NOT EXISTS table3 (
    col_1   STRING,
    col_2   int
)
USING parquet
PARTITIONED BY (col_2)
OPTIONS (
    path 'obs://bucketName/filePath',
    multiLeveldirenable = true,
    datadelegated = true,
    compression = 'zstd'
);
```

示例 6：创建 OBS 非分区表，自定义表的 OPTIONS 参数

示例说明：CSV表是一种以逗号分隔的纯文本文件格式，用于存储和交换数据。它通常用于简单的数据交换，但是它没有结构化数据的概念，因此不适合存储复杂数据类型。于是当file_format为csv时，支持配置更多的OPTIONS参数（参考**表1-10**）。

本例创建一个名为table4且存储格式为csv非分区表并使用了额外的OPTIONS参数对数据加以约束。

- delimiter：数据分隔符，表示使用逗号（，）作为数据之间的分隔符；
- quote：引用字符，表示使用双引号（”）来表示数据中的引用信息；
- escape：转义字符，表示使用反斜杠（\）作为数据存储时的转义字符；
- multiLine：设置需要存储的列数据中不包含回车符或者换行符；

- dataFormat：表示该csv文件中data字段的指定日期格式为yyyy-MM-dd；
- timestamoFormat：表示该csv文件中会将时间戳格式指定为yyyy-MM-dd HH:mm:ss；
- header：表示该csv表中包含表头信息；
- nullValue：表示设置null来表示csv表中的null值；
- comment：表示该csv表使用斜杠（/）表示注释的开头；
- compression：表示该csv表被压缩，此处csv表支持gzip、bzip2和deflate的压缩格式，若不需要压缩，也可以设置为none；
- encoding：表示该表使用utf-8的数据编码格式，在实际使用中，可以根据您的需求选择utf-8、gb2312和gbk中任一种编码格式，其中默认编码格式为utf-8。

```
CREATE TABLE IF NOT EXISTS table4 (
   col_1 STRING,
   col_2 INT
)
USING csv
OPTIONS (
   path 'obs://bucketName/filePath',
   delimiter       = ',',
   quote           = '#',
   escape          = '|',
   multiline       = false,
   dateFormat      = 'yyyy-MM-dd',
   timestampFormat  = 'yyyy-MM-dd HH:mm:ss',
   mode            = 'failfast',
   header          = true,
   nullValue       = 'null',
   comment         = '*',
   compression     = 'deflate',
   encoding        = 'utf - 8'
);
```

1.5.2 使用 Hive 语法创建 OBS 表

功能描述

使用Hive语法创建OBS表。DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

说明

推荐使用OBS并行文件系统进行存储。并行文件系统是一种高性能文件系统，提供毫秒级别访问时延，TB/s级别带宽和百万级别的IOPS，适用于大数据交互式分析场景。

注意事项

- 创建表时会统计大小。
- 添加数据时不会修改大小。
- 如需查看表大小可以通过OBS查看。
- CTAS建表语句不能指定表的属性。
- **关于分区表的使用说明：**
  - 创建分区表时，PARTITONED BY中指定分区列必须是不在表中的列，且需要指定数据类型。分区列支持string, boolean, tinyint, smallint, short, int, bigint, long, decimal, float, double, date, timestamp等hive开源支持的类型。

  - 支持指定多个分区字段，分区字段只需在PARTITIONED BY关键字后指定，不能像普通字段一样在表名后指定，否则将出错。
  - 单表分区数最多允许200000个。
  - CTAS建表语句不支持创建分区表。
- **关于创建表时设置多字符的分隔符：**
  - 只有指定ROW FORMAT SERDE为org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe时，字段分隔符才支持设置为多字符。
  - 只有Hive OBS表支持在建表时指定多字符的分隔符，Hive DLI表不支持在建表时指定多字符的分隔符。
  - 指定了多字符分隔的表不支持INSERT、IMPORT等写数语句。如需添加数据，请将数据文件直接放到表对应的OBS路径下即可，例如**示例7：创建表并设置多字符的分隔符**中，将数据文件放到obs://bucketName/filePath下。

语法格式

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name2 col_type2, [COMMENT col_comment2], ...)]
  [ROW FORMAT row_format]
  [STORED AS file_format]
  LOCATION 'obs_path'
  [TBLPROPERTIES (key = value)]
  [AS select_statement]
row_format:
  : SERDE serde_cls [WITH SERDEPROPERTIES (key1=val1, key2=val2, ...)]
  | 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]
```

关键字

- EXTERNAL：指创建OBS表。
- IF NOT EXISTS：指定该关键字以避免表已经存在时报错。
- COMMENT：字段或表描述。
- PARTITIONED BY：指定分区字段。
- ROW FORMAT：行数据格式。
- STORED AS：指定所存储的文件格式，当前该关键字只支持指定TEXTFILE, AVRO, ORC, SEQUENCEFILE, RCFILE, PARQUET格式。
- LOCATION：指定OBS的路径。创建OBS表时必须指定此关键字。
- TBLPROPERTIES：TBLPROPERTIES子句允许用户给表添加key/value的属性。
- AS：使用CTAS创建表。
- ROW FORMAT SERDE为org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe时，字段分隔符才支持设置为多字符。使用方法参考**示例7：创建表并设置多字符的分隔符**。

参数说明

表 1-11 参数说明

参数	是否必选	描述
db_name	否	Database名称。 由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。
table_name	是	Database中的表名。 由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。 特殊字符需要使用单引号（''）包围起来。 表名对大小写不敏感，即不区分大小写。
col_name	是	列字段名称。 列字段由字母、数字和下划线（_）组成。不能是纯数字，且至少包含一个字母。 列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。
col_comment	否	列字段描述。仅支持字符串常量。
row_format	是	行数据格式。row_format功能只支持textfile类型的表。
file_format	是	OBS表存储格式，支持TEXTFILE, AVRO, ORC, SEQUENCEFILE, RCFILE, PARQUET
table_comment	否	表描述。仅支持字符串常量。
obs_path	是	数据文件所在的OBS存储路径，推荐使用OBS并行文件系统存储。 格式：obs://bucketName/tblPath bucketName即桶名称。 tblPath是目录名称。目录后不需要指定文件名。 当OBS的目录下文件夹与文件同名时，创建OBS表指向的路径会优先指向文件而非文件夹。
key = value	否	设置TBLPROPERTIES具体属性和值。
select_statement	否	用于CTAS命令，将源表的select查询结果或某条数据插入到新创建的OBS表中。

示例 1：创建 OBS 非分区表

示例说明：创建名为table1的OBS非分区表，并用STORED AS关键字指定该表的存储格式为orc格式。

在您的实际使用中，可以将OBS表存储为textfile, avro, orc, sequencefile, rcfile, parquet等类型。

```
CREATE TABLE IF NOT EXISTS table1 (
   col_1   STRING,
   col_2   INT
)
STORED AS orc
LOCATION 'obs://bucketName/filePath';
```

示例 2：创建 OBS 分区表

示例说明：创建一个名为student的分区表，该分区表使用院系编号（facultyNo）和班级编号（classNo）进行分区，该student表会同时按照不同的院系编号（facultyNo）和不同的班级编号（classNo）分区。

在实际的使用过程中，您可以选择合适的分区字段并将其添加到PARTITIONED BY关键字后。

```
CREATE TABLE IF NOT EXISTS student(
   id      INT,
   name    STRING
)
STORED AS avro
LOCATION 'obs://bucketName/filePath'
PARTITIONED BY (
   facultyNo   INT,
   classNo     INT
);
```

示例 3：使用 CTAS 语句将源表的全部数据或部分数据创建新的 OBS 表

示例说明：根据**示例1：创建OBS非分区表**中创建的OBS表table1，使用CTAS语法将table1中的数据复制到table1_ctas表中。

在使用CTAS建表的时候，可以忽略被复制的表在建表时所使用的语法，即不论在创建table1时使用的是何种语法，都可以使用DataSource语法的CTAS创建table1_ctas。

此外，本例中table1中OBS表的存储格式为orc，而table1_ctas表的存储格式可以为sequencefile或者parquet，即CTAS创建的表存储格式可以不同于原表。

在AS关键字后使用SELECT语句选择需要的数据插入到table1_ctas表中。

SELECT语法为：SELECT <列名称> FROM <表名称> WHERE <相关筛选条件>。

- 示例中使用“SELECT * FROM table1”，'*'表示会从table1中选择所有列，并将table1中所有数据插入到table1_ctas表中。

  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  STORED AS sequencefile
  LOCATION 'obs://bucketName/filePath'
  AS
  SELECT  *
  FROM    table1;
  ```

- 若需要按照自定义方式筛选数据插入table1_ctas中，可以使用如下的SELECT语句“SELECT col_1 FROM table1 WHERE col_1 = 'Ann' ”，这样就可以通过执行SELECT语句从table1中单独选定col_1，并只将其中值等于'Ann'的数据插入到table1_ctas中。

  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  STORED AS parquet
  LOCATION  'obs:// bucketName/filePath'
  AS
  SELECT  col_1
  ```



```
FROM    table1
WHERE   col_1 = 'Ann';
```

示例 4：创建 OBS 非分区表，并自定义列字段数据类型

示例说明：创建名为table2的OBS非分区表，您可以根据业务需求自定义列字段的原生数据类型：

- 与文字字符有关可以使用STRING、CHAR或者VARCHAR。
- 与时间有关的可以使用TIMESTAMP、DATE。
- 与整数有关的可以使用INT、SMALLINT/SHORT、BIGINT/LONG、TINYINT。
- 涉及小数运算可以使用FLOAT、DOUBLE、DECIMAL。
- 若数据只涉及逻辑开关可以使用BOOLEAN类型。

具体使用方法与明细可以参照“数据类型 >原生数据类型”。

```
CREATE TABLE IF NOT EXISTS table2 (
   col_01  STRING,
   col_02  CHAR (2),
   col_03  VARCHAR (32),
   col_04  TIMESTAMP,
   col_05  DATE,
   col_06  INT,
   col_07  SMALLINT,
   col_08  BIGINT,
   col_09  TINYINT,
   col_10  FLOAT,
   col_11  DOUBLE,
   col_12  DECIMAL (10, 3),
   col_13  BOOLEAN
)
STORED AS parquet
LOCATION 'obs://bucketName/filePath';
```

示例 5：创建 OBS 分区表，自定义表的 TBLPROPERTIES 参数

示例说明：创建名为table3，并以col_3为分区依据的OBS分区表。在TBLPROPERTIES中配置dli.multi.version.enable、comment、orc.compress和auto.purge。

- dli.multi.version.enable：本例配置为true，即代表开启DLI数据多版本功能，用于表数据的备份与恢复。
- comment：表描述信息，comment描述信息支持后续修改。
- orc.compress：指定orc存储的压缩方式，本例定义为ZLIB。
- auto.purge：本例配置为true，即删除或者覆盖的数据会不经过回收站，直接被删除。

```
CREATE TABLE IF NOT EXISTs table3 (
   col_1 STRING,
   col_2 STRING
)
PARTITIONED BY (col_3 DATE)
STORED AS rcfile
LOCATION 'obs://bucketName/filePath'
TBLPROPERTIES (
   dli.multi.version.enable  = true,
   comment                   = 'Created by dli',
   orc.compress              = 'ZLIB',
   auto.purge                = true
);
```


示例 6：创建 textfile 格式的非分区表，并设置 ROW FORMAT

示例说明：创建名为table4的textfile类型的非分区表，并设置ROW FORMAT（ ROW FORMAT功能只支持textfile类型的表 ）。

- FIELDS：字段表格中的列，每个字段有一个名称和数据类型，表中字段之间以'/'分隔。
- COLLECTION ITEMS：集合项指的是一组数据中的元素，可以是数组、列表或集合等，表中集合项以'$'分隔。
- MAP KEYS：映射键是一种键值对的数据结构，用于存储一组相关联的数据，表中Map键以'#'分隔。
- LINES：表格中的行，每一行包含一组字段值，表中行以'\n'结束（ 注意，只支持用'\n'作为行分隔符 ）。
- NULL：表示缺少值或未知值的特殊值。在表格中，NULL表示该字段没有值或该值未知。如果数据中存在null值，则用字符串“null”表示。

```
CREATE TABLE IF NOT EXISTS table4 (
   col_1   STRING,
   col_2   INT
)
STORED AS textfile
LOCATION 'obs://bucketName/filePath'
ROW FORMAT
DELIMITED FIELDS TERMINATED    BY '/'
COLLECTION ITEMS TERMINATED    BY '$'
MAP KEYS TERMINATED            BY '#'
LINES TERMINATED               BY '\n'
NULL DEFINED                  AS 'null';
```

示例 7：创建表并设置多字符的分隔符

示例说明：创建了一个名为table5的Hive表。表指定序列化和反序列化类ROW FORMAT SERDE，字段之间的分隔符被设置为/#，并且数据以文本文件格式存储。

- 只有指定ROW FORMAT SERDE为org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe时，字段分隔符才支持设置为多字符。
- 只有Hive OBS表支持在建表时指定多字符的分隔符，Hive DLI表不支持在建表时指定多字符的分隔符。
- 指定了多字符分隔的表不支持INSERT、IMPORT等写数语句。如需添加数据，请将数据文件直接放到表对应的OBS路径下即可，本例中，将数据文件放到obs://bucketName/filePath下。
- 本例指定字段分隔符 field.delim'为“/#”。
- ROW FORMAT功能只支持textfile类型的表。

```
CREATE TABLE IF NOT EXISTS table5 (
   col_1   STRING,
   col_2   INT
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe'
WITH SERDEPROPERTIES (
 'field.delim' = '/#'
)
STORED AS textfile
LOCATION 'obs://bucketName/filePath';
```

1.6 创建 DLI 表

1.6.1 使用 DataSource 语法创建 DLI 表

功能描述

使用DataSource语法创建DLI表。DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

注意事项

- CTAS建表语句不能指定表的属性。
- 若没有指定分隔符，则默认为逗号（,）。
- **关于分区表的使用说明：**
  - 创建分区表时，PARTITIONED BY中指定分区列必须是表中的列，且必须在Column列表中指定类型。分区列只支持string, boolean, tinyint, smallint, short, int, bigint, long, decimal, float, double, date, timestamp类型。
  - 创建分区表时，分区字段必须是表字段的最后一个字段或几个字段，且多分区字段的顺序也必须对应。否则将出错。
  - 单表分区数最多允许200000个。
  - CTAS建表语句不支持创建分区表。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  USING file_format
  [OPTIONS (key1=val1, key2=val2, ...)]
  [PARTITIONED BY (col_name1, col_name2, ...)]
  [COMMENT table_comment]
  [AS select_statement];
```

关键字

- IF NOT EXISTS：指定该关键字以避免表已经存在时报错。
- USING：指定存储格式。
- OPTIONS：指定建表时的属性名与属性值。
- COMMENT：字段或表描述。
- PARTITIONED BY：指定分区字段。
- AS：使用CTAS创建表。

参数说明

表 1-12 参数描述

参数	是否必选	描述
db_name	否	Database名称。 由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以数字和下划线开头。
table_name	是	Database中的表名。 由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以数字和下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。 特殊字符需要使用单引号（ '' ）包围起来。 表名对大小写不敏感，即不区分大小写。
col_name	是	以逗号分隔的带数据类型的列名。 列名由字母、数字和下划线（ _ ）组成。不能是纯数字，且至少包含一个字母。 列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。
col_comment	否	列字段描述。仅支持字符串常量。
file_format	是	DLI表数据存储格式，支持：parquet和orc格式。
table_comment	否	表描述。仅支持字符串常量。
select_statement	否	用于CTAS命令，将源表的select查询结果或某条数据插入到新创建的DLI表中。

表 1-13 OPTIONS 参数描述

参数	是否必选	描述	默认值
multiLevelDirEnable	否	是否迭代查询子目录中的数据。当配置为true时，查询该表时会迭代读取该表路径中所有文件，包含子目录中的文件。	false
compression	否	指定压缩格式。一般为parquet格式时指定该参数，推荐使用'zstd'压缩格式。	-

示例 1：创建 DLI 非分区表

示例说明：创建名为table1的DLI非分区表，使用USING关键字指定该表的存储格式为orc格式。

在您的实际使用中，还可以将DLI表存储为parquet类型。

```
CREATE TABLE IF NOT EXISTS table1 (
  col_1 STRING,
  col_2 INT)
USING orc;
```

示例 2：创建 DLI 分区表

示例说明：创建一个名为student的分区表，该分区表使用院系编号（ facultyNo ）和班级编号（ classNo ）进行分区，该student表会同时按照不同的院系编号（ facultyNo ）和不同的班级编号（ classNo ）分区。

在实际的使用过程中，您可以选择合适的分区字段并将其添加到PARTITIONED BY关键字后。

```
CREATE TABLE IF NOT EXISTS student (
  Name       STRING,
  facultyNo  INT,
  classNo    INT
)
USING orc
PARTITIONED BY (facultyNo, classNo);
```

示例 3：使用 CTAS 将源表的全部数据或部分数据创建新的 DLI 表

示例说明：根据**示例1：创建DLI非分区表**中创建的DLI表table1，使用CTAS语法将table1中的数据复制到table1_ctas表中。

在使用CTAS建表的时候，可以忽略被复制的表在建表时所使用的语法，即不论在创建table1时使用的是何种语法，都可以使用DataSource语法的CTAS创建table1_ctas。

此外，本例中table1DLI表的存储格式为orc，而table1_ctas表的存储格式可以为orc或者parquet，即CTAS创建的表存储格式可以不同于原表。

在AS关键字后使用SELECT语句选择需要的数据插入到table1_ctas表中。

SELECT语法为：SELECT <列名称> FROM <表名称> WHERE <相关筛选条件>。

- 示例中使用“SELECT * FROM table1”，'*'表示会从table1中选择所有列，并将table1中所有数据插入到table1_ctas表中。
  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  USING parquet
  AS
  SELECT  *
  FROM    table1;
  ```
- 若需要按照自定义方式筛选数据插入table1_ctas中，可以使用如下的SELECT语句“SELECT col_1 FROM table1 WHERE col_1 = 'Ann' ”，这样就可以通过执行SELECT语句从table1中单独选定col_1，并只将其中值等于'Ann'的数据插入到table1_ctas中。
  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  USING parquet
  AS
  SELECT  col_1
  FROM    table1
  WHERE   col_1 = 'Ann';
  ```


示例 4：创建 DLI 非分区表，并自定义列字段数据类型

示例说明：创建名为table2的DLI非分区表，您可以根据业务需求自定义列子段的原生数据类型：

- 与文字字符有关可以使用STRING、CHAR或者VARCHAR。
- 与时间有关的可以使用TIMESTAMP、DATE。
- 与整数有关的可以使用INT、SMALLINT/SHORT、BIGINT/LONG、TINYINT。
- 涉及小数运算可以使用FLOAT、DOUBLE、DECIMAL。
- 若数据只涉及逻辑开关可以使用BOOLEAN类型。

具体使用方法与明细可以参照“数据类型 >原生数据类型”。

```
CREATE TABLE IF NOT EXISTS table2 (
   col_01  STRING,
   col_02  CHAR (2),
   col_03  VARCHAR (32),
   col_04  TIMESTAMP,
   col_05  DATE,
   col_06  INT,
   col_07  SMALLINT,
   col_08  BIGINT,
   col_09  TINYINT,
   col_10  FLOAT,
   col_11  DOUBLE,
   col_12  DECIMAL (10, 3),
   col_13  BOOLEAN
)
USING parquet;
```

示例 5：创建 DLI 分区表，自定义表的 OPTIONS 参数

示例说明：创建DLI表时支持自定义属性名与属性值，OPTIONS参数说明可参考**表1-13**。

本例创建名为table3并以col_2为分区依据的DLI分区表。在OPTIONS中配置pmultiLevelDirEnable和compression。

- multiLevelDirEnable：本例设置为true，表示查询该表时会迭代读取表路径中的所有文件和子目录文件，若不需要此项配置可以设置为false或不设置（默认为false）；
- compression：当创建的OBS表需要压缩时，可以使用compression关键字来配置压缩格式，本例中就使用了zstd压缩格式。

```
CREATE TABLE IF NOT EXISTs table3 (
   col_1   STRING,
   col_2   int
)
USING parquet
PARTITIONED BY (col_2)
OPTIONS (
   multiLeveldirenable    = true,
   compression            = 'zstd'
);
```

1.6.2 使用 Hive 语法创建 DLI 表

功能描述

使用Hive语法创建DLI表。DataSource语法和Hive语法主要区别在于支持的表数据存储格式范围、支持的分区数等有差异，详细请参考语法格式和注意事项说明。

注意事项

- CTAS建表语句不能指定表的属性。
- Hive DLI表不支持在建表时指定多字符的分隔符。
- **关于分区表的使用说明：**
  - 创建分区表时，PARTITONED BY中指定分区列必须是不在表中的列，且需要指定数据类型。分区列支持string, boolean, tinyint, smallint, short, int, bigint, long, decimal, float, double, date, timestamp等hive开源支持的类型。
  - 支持指定多个分区字段，分区字段只需在PARTITIONED BY关键字后指定，不能像普通字段一样在表名后指定，否则将出错。
  - 单表分区数最多允许200000个。
  - CTAS建表语句不支持创建分区表。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name1 col_type1 [COMMENT col_comment1], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name2 col_type2, [COMMENT col_comment2], ...)]
  [ROW FORMAT row_format]
  STORED AS file_format
  [TBLPROPERTIES (key = value)]
  [AS select_statement];

row_format:
  : SERDE serde_cls [WITH SERDEPROPERTIES (key1=val1, key2=val2, ...)]
  | 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]
```

关键字

- IF NOT EXISTS：指定该关键字以避免表已经存在时报错。
- COMMENT：字段或表描述。
- PARTITIONED BY：指定分区字段。
- ROW FORMAT：行数据格式。
- STORED AS：指定所存储的文件格式，当前该关键字只支持指定TEXTFILE, AVRO, ORC, SEQUENCEFILE, RCFILE, PARQUET几种格式。创建DLI表时必须指定此关键字。
- TBLPROPERTIES：用于为表添加key/value的属性。
  - 在表存储格式为PARQUET时，可以通过指定TBLPROPERTIES(parquet.compression = 'zstd')来指定表压缩格式为zstd。
- AS：使用CTAS创建表。

参数说明

表 1-14 参数描述

参数	是否必选	描述
db_name	否	Database名称。 由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。
table_name	是	Database中的表名。 由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（''）包围起来。
col_name	是	列字段名称。 列字段由字母、数字和下划线（_）组成。不能是纯数字，且至少包含一个字母。 列名为大小写不敏感，即不区分大小写。
col_type	是	列字段的数据类型。数据类型为原生类型。
col_comment	否	列字段描述。仅支持字符串常量。
row_format	是	行数据格式。row format功能只支持textfile类型的表。
file_format	是	DLI表数据存储格式：支持textfile, avro, orc, sequencefile, rcfile, parquet。
table_comment	否	表描述。仅支持字符串常量。
key = value	否	设置TBLPROPERTIES具体属性和值。 在表存储格式为PARQUET时，可以通过指定TBLPROPERTIES(parquet.compression = 'zstd')来指定表压缩格式为zstd。
select_statement	否	用于CTAS命令，将源表的select查询结果或某条数据插入到新创建的DLI表中。

示例 1：创建 DLI 非分区表

示例说明：创建名为table1的DLI非分区表，并用STORED AS关键字指定该表的存储格式为orc格式。

在您的实际使用中，可以将DLI表存储为textfile, avro, orc, sequencefile, rcfile, parquet等类型。

```
CREATE TABLE IF NOT EXISTS table1 (
   col_1   STRING,
   col_2   INT
)
STORED AS orc;
```

示例 2：创建 DLI 分区表

示例说明：创建一个名为student的分区表，该分区表使用院系编号（facultyNo）和班级编号（classNo）进行分区，该student表会同时按照不同的院系编号（facultyNo）和不同的班级编号（classNo）分区。

在实际的使用过程中，您可以选择合适的分区字段并将其添加到PARTITIONED BY关键字后。

```
CREATE TABLE IF NOT EXISTS student(
    id      int,
    name    STRING
)
STORED AS avro
PARTITIONED BY (
    facultyNo   INT,
    classNo     INT
);
```

示例 3：使用 CTAS 语句将源表的全部数据或部分数据创建新的 DLI 表

示例说明：根据**示例1：创建DLI非分区表**中创建的DLI表table1，使用CTAS语法将table1中的数据复制到table1_ctas表中。

在使用CTAS建表的时候，可以忽略被复制的表在建表时所使用的语法，即不论在创建table1时使用的是何种语法，都可以使用DataSource语法的CTAS创建table1_ctas。

本例中table1中DLI表的存储格式为orc，而table1_ctas表的存储格式可以为parquet，即CTAS创建的表存储格式可以不同于原表。

在AS关键字后使用select语句选择需要插入到table1_ctas表中的数据。

SELECT语法为：SELECT <列名称> FROM <表名称> WHERE <相关筛选条件>。

- 示例中使用select * from table1，表示会从table1中选择所有语句，并将这些语句复制到table1_ctas表中。

  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  STORED AS sequencefile
  AS
  SELECT  *
  FROM table1;
  ```

- 若不需要table1中的全部数据，可以将“AS SELECT * FROM table1”改为“AS SELECT col_1 FROM table1 WHERE col_1 = “Ann”，这样就可以通过执行SELECT语句从table1中单独指定col_1列等于'Ann'的所有行插入到table1_ctas中。

  ```
  CREATE TABLE IF NOT EXISTS table1_ctas
  USING parquet
  AS
  SELECT col_1
  FROM table1
  WHERE col_1 = 'Ann';
  ```

示例 4：创建 DLI 非分区表，并自定义列字段数据类型

示例说明：创建名为table2的DLI非分区表，您可以根据业务需求自定义列子段的原生数据类型：

- 与文字字符有关可以使用STRING、CHAR或者VARCHAR。
- 与时间有关的可以使用TIMESTAMP、DATE。

- 与整数有关的可以使用INT、SMALLINT/SHORT、BIGINT/LONG、TINYINT。
- 涉及小数运算可以使用FLOAT、DOUBLE、DECIMAL。
- 若数据只涉及逻辑开关可以使用BOOLEAN类型。

具体使用方法与明细可以参照“数据类型 >原生数据类型”。

```
CREATE TABLE IF NOT EXISTS table2 (
   col_01  STRING,
   col_02  CHAR (2),
   col_03  VARCHAR (32),
   col_04  TIMESTAMP,
   col_05  DATE,
   col_06  INT,
   col_07  SMALLINT,
   col_08  BIGINT,
   col_09  TINYINT,
   col_10  FLOAT,
   col_11  DOUBLE,
   col_12  DECIMAL (10, 3),
   col_13  BOOLEAN
)
STORED AS parquet;
```

示例 5：创建 DLI 分区表，自定义表的 TBLPROPERTIES 参数

示例说明：本例创建名为table3并以col_3为分区依据的DLI分区表。在TBLPROPERTIES中配置dli.multi.version.enable、comment、orc.compress和auto.purge。

- dli.multi.version.enable：本例配置为true，即代表开启DLI数据多版本功能，用于表数据的备份与恢复。
- comment：表描述信息，TBLPROPERTIES内的描述信息支持后续修改。
- orc.compress：指定orc存储的压缩方式，本例定义为ZLIB。
- auto.purge：本例配置为true，即删除或者覆盖的数据会不经过回收站，直接被删除。

```
CREATE TABLE IF NOT EXISTs table3 (
   col_1 STRING,
   col_2 STRING
)
PARTITIONED BY (col_3 DATE)
STORED AS rcfile
TBLPROPERTIES (
   dli.multi.version.enable    = true,
   comment                     = 'Created by dli',
   orc.compress                = 'ZLIB',
   auto.purge                  = true
);
```

示例 6：创建 textfile 格式的非分区表，并设置 ROW FORMAT

示例说明：本例创建名为table4的textfile类型的非分区表，并设置ROW FORMAT相关格式（ ROW FORMAT功能只支持textfile类型的表 ）。

- 字段（ Fields ）是表格中的列，每个字段有一个名称和数据类型，表中字段之间以'/'分隔。
- 集合项（ COLLECTION ITEMS ）指的是一组数据中的元素，可以是数组、列表或集合等，table4中集合项以'$'分隔。
- 映射键（ MAP KEYS ）是一种键值对的数据结构，用于存储一组相关联的数据，表中Map键以'#'分隔。

- 行（Rows）表格中的行，每一行包含一组字段值，表中行以'\n'结束（注意，只支持用'\n'作为行分隔符）。
- NULL表示缺少值或未知值的特殊值。在表格中，NULL表示该字段没有值或该值未知。如果数据中存在null值，则用字符串“null”表示。

```
CREATE TABLE IF NOT EXISTS table4 (
   col_1   STRING,
   col_2   INT
)
STORED AS TEXTFILE
ROW FORMAT
DELIMITED FIELDS TERMINATED   BY '/'
COLLECTION ITEMS TERMINATED   BY '$'
MAP KEYS TERMINATED           BY '#'
LINES TERMINATED              BY '\n'
NULL DEFINED                 AS 'NULL';
```

1.7 删除表

功能描述

删除表。

语法格式

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

关键字

- OBS表：仅删除其元数据信息，不删除存放在OBS上的数据。

参数说明

表 1-15 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。
table_name	表名称。

注意事项

所要删除的表必须是当前数据库下存在的，否则会出错，可以通过添加IF EXISTS来避免出错。

示例

1. 参考**创建OBS表**中的示例描述创建对应的表。
2. 在当前所在数据库下删除名为test的表。
   ```
   DROP TABLE IF EXISTS test;
   ```

1.8 查看表

1.8.1 查看所有表

功能描述

查看当前数据库下所有的表。显示当前数据库下的所有表及视图。

语法格式

```
SHOW TABLES [IN | FROM db_name] [LIKE regex_expression];
```

关键字

FROM/IN：指定数据库名，显示特定数据库下的表及视图。

参数说明

表 1-16 参数说明

参数	描述
db_name	数据库名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以数字和下划线开头。
regex_expression	数据库下的表名称。

注意事项

无。

示例

1. 参考**创建OBS表**中的示例描述创建对应的表。
2. 查看当前所在数据库中的所有表与视图。
   ```
   SHOW TABLES;
   ```
3. 查看testdb数据库下所有以test开头的表。
   ```
   SHOW TABLES IN testdb LIKE "test*";
   ```

1.8.2 查看建表语句

功能描述

返回对应表的建表语句。

语法格式

```
SHOW CREATE TABLE table_name;
```

关键字

CREATE TABLE：建表语句。

参数说明

表 1-17 参数说明

参数	描述
table_name	表名称。

注意事项

语句所涉及的表必须存在，否则会出错。

示例

Saprk 2.4.5版本示例：

- 执行以下命令返回测试表testDB01.testTable5的建表语句

  SHOW CREATE TABLE testDB01.testTable5

- 返回test表的建表语句：

```
createtab_stmt
 CREATE TABLE `testDB01`.`testTable5`(`id` INT, `age` INT, `money` DOUBLE)
COMMENT 'test'
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = '1'
)
STORED AS
  INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
  OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
TBLPROPERTIES (
  'hive.serialization.extend.nesting.levels' = 'true',
  'ddlUpdateTime' = '1707202585460'
)
```

1.8.3 查看表属性

功能描述

查看表的属性。

语法格式

```
SHOW TBLPROPERTIES table_name [('property_name')];
```

关键字

TBLPROPERTIES：TBLPROPERTIES子句允许用户给表添加key/value的属性。

参数说明

表 1-18 参数说明

参数	描述
table_name	表名称。
property_name	• 命令中不指定property_name时，将返回所有属性及其值； • 命令中指定property_name时，将返回该特定property_name所对应的值。

注意事项

property_name大小写敏感，不能同时指定多个property_name，否则会出错。

示例

返回test表中属性property_key1的值。

```
SHOW TBLPROPERTIES test ('property_key1');
```

1.8.4 查看指定表所有列

功能描述

查看指定表中的所有列。

语法格式

```
SHOW COLUMNS {FROM | IN} table_name [{FROM | IN} db_name];
```

关键字

- COLUMNS：表中的列。
- FROM/IN：指定数据库，显示指定数据库下的表的列名。FROM和IN没有区别，可替换使用。

参数说明

表 1-19 参数说明

参数	描述
table_name	表名称。
db_name	数据库名称。

注意事项

所指定的表必须是数据库中存在的表，否则会出错。

示例

查看student表中的所有列。

```
SHOW COLUMNS IN student;
```

1.8.5 查看指定表所有分区

功能描述

查看指定表的所有分区。

语法格式

```
SHOW PARTITIONS [db_name.]table_name
  [PARTITION partition_specs];
```

关键字

- PARTITIONS：表中的分区。
- PARTITION：分区。

参数说明

表 1-20 参数描述

参数	描述
db_name	Database名称，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以下划线开头。
table_name	Database中的表名，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以下划线开头。 匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$，如果特殊字符需要使用单引号（ '' ）包围起来。
partition_specs	分区信息，key=value形式，key为分区字段，value为分区值。若分区字段为多个字段，可以不包含所有的字段，会显示匹配上的所有分区信息。

注意事项

所要查看分区的表必须存在且是分区表，否则会出错。

示例

- 查看student表下面的所有的分区。
  ```
  SHOW PARTITIONS student;
  ```

- 查看student表中dt='2010-10-10'的分区。
  ```
  SHOW PARTITIONS student PARTITION(dt='2010-10-10');
  ```

1.8.6 查看表统计信息

功能描述

查看表统计信息。返回所有列的列名和列数据类型。

语法格式

```
DESCRIBE [EXTENDED|FORMATTED] [db_name.]table_name;
```

关键字

- EXTENDED：显示表的所有元数据，通常只在debug时用到。
- FORMATTED：使用表格形式显示所有表的元数据。

参数说明

表 1-21 参数描述

参数	描述
db_name	Database名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。
table_name	Database中的表名，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（''）包围起来。

注意事项

若所查看的表不存在，将会出错。

示例

查看student表的所有列的列名与列数据类型。

```
DESCRIBE student;
```

1.9 修改表

1.9.1 添加列

功能描述

添加一个或多个新列到表上。

语法格式

```
ALTER TABLE [db_name.]table_name ADD COLUMNS (col_name1 col_type1 [COMMENT
col_comment1], ...);
```

关键字

- ADD COLUMNS：添加列。
- COMMENT：列描述。

参数说明

表 1-22 参数描述

参数	描述
db_name	Database名称，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以下划线开头。
table_name	表名称。
col_name	列字段名称。
col_type	列字段类型。
col_comment	列描述。

注意事项

该SQL不建议并发执行，并发情况下添加列结果可能互相覆盖。

示例

```
ALTER TABLE t1 ADD COLUMNS (column2 int, column3 string);
```

1.9.2 修改列注释

功能描述

修改非分区表或分区表的列注释信息。

语法格式

```
ALTER TABLE [db_name.]table_name CHANGE COLUMN col_name col_name col_type COMMENT
'col_comment';
```

关键字

- CHANGE COLUMN：修改列
- COMMENT：列描述。

参数说明

表 1-23 参数描述

参数	是否必选	描述
db_name	否	Database名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。
table_name	是	表名称。
col_name	是	列字段名称。col_name必须是已存在的列。
col_type	是	列数据类型。本语法不支持修改列数据类型，这里指定的是创建表时指定的列数据类型。
col_comment	是	修改后的列注释信息。注释内容为长度不超过1024字节的有效字符串。

示例

修改表t1中的c1列的注释信息为“the new comment”。

```
ALTER TABLE t1 CHANGE COLUMN c1 c1 STRING COMMENT 'the new comment';
```

1.10 分区表相关

1.10.1 添加分区（只支持 OBS 表）

功能描述

创建OBS分区表成功后，OBS表实际还没有生成分区信息。生成分区信息主要有以下两种场景：

- 给OBS分区表插入对应的分区数据，数据插入成功后OBS表才会生成分区元数据信息，后续则可以根据对应分区列进行查询等操作。
- 手工拷贝分区目录和数据到OBS分区表路径下，执行本章节介绍的分区添加命令生成分区元数据信息，后续即可根据对应分区列进行查询等操作。

本章节重点介绍使用**ALTER TABLE**命令添加分区的基本操作和使用说明。

语法格式

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION partition_specs1
  [LOCATION 'obs_path1']
  PARTITION partition_specs2
  [LOCATION 'obs_path2'];
```

关键字

- IF NOT EXISTS：指定该关键字以避免分区重复添加时报错。
- PARTITION：分区。
- LOCATION：分区路径。

参数说明

表 1-24 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。
obs_path	OBS存储路径。

注意事项

- 向表中添加分区时，此表和分区列（建表时PARTITIONED BY指定的列）必须已存在，而所要添加的分区不能重复添加，否则将出错。已添加的分区可通过IF NOT EXISTS避免报错。
- 若分区表是按照多个字段进行分区的，添加分区时需要指定所有的分区字段，指定字段的顺序可任意。
- “partition_specs”中的参数默认带有“( )”。例如：**PARTITION (dt='2009-09-09',city='xxx')**。
- 在添加分区时若指定OBS路径，则该OBS路径必须是已经存在的，否则会出错。
- 若添加多个分区，每组PARTITION partition_specs LOCATION 'obs_path'之间用空格隔开。例如：

  PARTITION partition_specs LOCATION 'obs_path' PARTITION partition_specs LOCATION 'obs_path'。
- 若新增分区指定的路径包含子目录（或嵌套子目录），则子目录下面的所有文件类型及内容也将作为该分区的记录。

  您需要保证该分区目录下所有文件类型和文件内容与表的字段一致，否则查询将报错。

  您可以在建表语句OPTIONS中设置“multiLevelDirEnable”为true以查询子目录下的内容，此参数默认值为false（注意，此配置项为表属性，请谨慎配置。Hive表不支持此配置项)。

示例

- 建OBS表时仅有**一个分区列**，建表成功后添加分区数据。

  a. 先使用DataSource语法创建一个OBS分区表，分区列为external_data，数据存储在obs://bucketName/datapath路径下。

  ```
  create table testobstable(id varchar(128), external_data varchar(16)) using JSON OPTIONS (path 'obs://bucketName/datapath') PARTITIONED by (external_data);
  ```


b. 拷贝分区数据目录到obs://bucketName/datapath路径下。例如当前拷贝external_data=22的分区目录下所有文件到obs://bucketName/datapath路径下。

c. 执行添加分区命令，将分区的元数据信息生效。

```
ALTER TABLE testobstable ADD
  PARTITION (external_data='22')
  LOCATION 'obs://bucketName/datapath/external_data=22';
```

d. 添加分区成功后，即可根据分区列进行数据查询等操作。

```
select * from testobstable where external_data='22';
```

- 建OBS表时有**多个分区列**，建表成功后添加分区数据。

a. 先使用DataSource语法创建一个OBS分区表，分区列为external_data和dt，数据存储在obs://bucketName/datapath路径下。

```
create table testobstable(
  id varchar(128),
  external_data varchar(16),
  dt varchar(16)
) using JSON OPTIONS (path 'obs://bucketName/datapath') PARTITIONED by (external_data,
dt);
```

b. 拷贝分区数据目录到obs://bucketName/datapath路径下。例如拷贝external_data=22及其子目录dt=2021-07-27和目录下文件到obs://bucketName/datapath路径下。

c. 执行添加分区命令，将分区的元数据信息生效。

```
ALTER TABLE
  testobstable
ADD
  PARTITION (external_data = '22', dt = '2021-07-27') LOCATION 'obs://bucketName/datapath/
external_data=22/dt=2021-07-27';
```

d. 添加分区成功后，即可根据分区列进行数据查询等操作。

```
select * from testobstable where external_data = '22';
select * from testobstable where external_data = '22' and dt='2021-07-27';
```

1.10.2 重命名分区（只支持 OBS 表）

功能描述

重命名分区。

语法格式

```
ALTER TABLE table_name
  PARTITION partition_specs
  RENAME TO PARTITION partition_specs;
```

关键字

- PARTITION：分区。
- RENAME：重命名。

参数说明

表 1-25 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。

注意事项

- **该命令仅支持操作OBS表，不支持对DLI表进行操作。**
- 所要重命名分区的表和分区必须已存在，否则会出错。新分区名不能与其他分区重名，否则将出错。
- 若分区表是按照多个字段进行分区的，重命名分区时需要指定所有的分区字段，指定字段的顺序可任意。
- “partition_specs”中的参数默认带有“( )”，例如：**PARTITION (dt='2009-09-09',city='xxx')**。

示例

将student表中的分区city='xxx',dt='2008-08-08'重命名为city='xxx',dt='2009-09-09'。

```
ALTER TABLE student
  PARTITION (city='xxx',dt='2008-08-08')
  RENAME TO PARTITION (city='xxx',dt='2009-09-09');
```

1.10.3 删除分区

功能描述

本节操作介绍删除分区表的一个或多个分区。

分区表分为两种，OBS表和DLI表。在删除分区时，DLI表和OBS表都支持利用指定条件删除分区表的一个或多个分区。OBS表还支持**按指定筛选条件删除分区**。

注意事项

- 所要删除分区的表必须是已经存在的表，否则会出错。
- 所要删除的分区必须是已经存在的，否则会出错，可通过语句中添加“IF EXISTS”避免该错误。

语法格式

```
ALTER TABLE [db_name.]table_name
  DROP [IF EXISTS]
  PARTITION partition_spec1[,PARTITION partition_spec2,...];
```

关键字

- DROP：删除表分区。
- IF EXISTS：所要删除的分区必须是已经存在的，否则会出错。
- PARTITION：分区。

参数说明

表 1-26 参数描述

参数	描述
db_name	Database名称，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以下划线开头。
table_name	Database中的表名，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（ '' ）包围起来。
partition_specs	分区信息，key=value形式，key为分区字段，value为分区值。若分区字段为多个字段，可以不包含所有的字段，会删除匹配上的所有分区。“partition_specs”中的参数默认带有“( )”，例如：**PARTITION (facultyNo=20, classNo=103);**。

示例

为了便于理解删除分区语句的使用方法，本节示例为您提供源数据，基于源数据提供删除分区的操作示例。

步骤1 使用DataSource语法创建一个OBS表分区表。

创建了一个名为student的OBS分区表，表中有学生学号（ id ），学生姓名（ name ），学生院系编号（ facultyNo ）和学生班级编号（ classNo ），该表使用学生院系编号（ facultyNo ）和学生班级编号（ classNo ）进行分区。

```
create table if not exists student (
id int,
name STRING,
facultyNo int,
classNo INT)
using csv
options (path 'obs://bucketName/filePath')
partitioned by (faculytNo, classNo);
```

步骤2 在表格中插入分区数据。

利用插入数据中的内容，可以插入以下数据

```
INSERT into student
partition (facultyNo = 10, classNo = 101)
values (1010101, "student01"), (1010102, "student02");

INSERT into student
partition (facultyNo = 10, classNo = 102)
values (1010203, "student03"), (1010204, "student04");

INSERT into student
partition (facultyNo = 20, classNo = 101)
```



```
values (2010105, "student05"), (2010106, "student06");

INSERT into student
partition (facultyNo = 20, classNo = 102)
values (2010207, "student07"), (2010208, "student08");

INSERT into student
partition (facultyNo = 20, classNo = 103)
values (2010309, "student09"), (2010310, "student10");

INSERT into student
partition (facultyNo = 30, classNo = 101)
values (3010111, "student11"), (3010112, "student12");

INSERT into student
partition (facultyNo = 30, classNo = 102)
values (3010213, "student13"), (3010214, "student14");
```

步骤3 查看分区。

利用查看指定表所有分区中的内容，可以查看相关的分区内容。

示例代码如下：

```
SHOW partitions student;
```

表 1-27 表数据示例

facultyNo	classNo
facultyNo=10	classNo=101
facultyNo=10	classNo=102
facultyNo=20	classNo=101
facultyNo=20	classNo=102
facultyNo=20	classNo=103
facultyNo=30	classNo=101
facultyNo=30	classNo=102

步骤4 删除分区。

- **示例1：指定多个筛选条件删除分区**

  本示例删除facultyNo为20，classNo为103的分区；

  说明

  如需按指定筛选条件删除分区请参考**指定筛选条件删除分区（只支持OBS表）**。

  示例代码如下：

  ```
  ALTER TABLE student
  DROP IF EXISTS
  PARTITION (facultyNo=20, classNo=103);
  ```

  重新利用第三步中的方法查看表中的分区，可以看到该分区被删除：

  ```
  SHOW partitions student;
  ```

- **示例2：指定单个筛选条件删除分区**

本示例删除facultyNo为30的分区；在插入数据的过程中可以了解到，facultyNo为30的分区有两个。

说明

如需按指定筛选条件删除分区请参考**指定筛选条件删除分区（只支持OBS表）**。

示例代码如下：

```
ALTER TABLE student
DROP IF EXISTS
PARTITION (facultyNo = 30);
```

执行后结果：

表 1-28 表数据示例

facultyNo	classNo
facultyNo=10	classNo=101
facultyNo=10	classNo=102
facultyNo=20	classNo=101
facultyNo=20	classNo=102
facultyNo=20	classNo=103

----结束

1.10.4 指定筛选条件删除分区（只支持 OBS 表）

功能描述

指定筛选条件删除分区表的一个或多个分区。

注意事项

- 该命令仅支持操作OBS表，不支持对DLI表进行操作。
- 所要删除分区的表必须是已经存在的表，否则会出错。
- 所要删除的分区必须是已经存在的，否则会出错，可通过语句中添加“IF EXISTS”避免该错误。

语法格式

```
ALTER TABLE [db_name.]table_name
  DROP [IF EXISTS]
  PARTITIONS partition_filtercondition;
```

关键字

- DROP：删除表分区。
- IF EXISTS：所要删除的分区必须是已经存在的，否则会出错。
- PARTITIONS：分区。

参数说明

表 1-29 参数描述

参数	描述
db_name	Database名称，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以下划线开头。
table_name	Database中的表名，由字母、数字和下划线（ _ ）组成。不能是纯数字，且不能以下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（ '' ）包围起来。 **该命令仅支持操作OBS表，不支持对DLI表进行操作。**
partition_filtercondition	分区筛选条件。具体可以为以下格式： <分区列名> <**运算符**> <分区列比较值> 例如：start_date < '201911' • 示例1：<partition_filtercondition1> AND|OR <partition_filtercondition2> 例如：start_date < '201911' OR start_date >= '202006' • 示例2：（<partition_filtercondition1>)[,partitions (<partition_filtercondition2>), ...] 例如：(start_date <> '202007'), partitions(start_date < '201912')

示例

为了便于理解删除分区语句的使用方法，本节示例为您提供源数据，基于源数据提供删除分区的操作示例。

步骤1 使用DataSource语法创建一个OBS表分区表。

创建了一个名为student的OBS分区表，表中有学生学号（ id ），学生姓名（ name ），学生院系编号（ facultyNo ）和学生班级编号（ classNo ），该表使用学生院系编号（ facultyNo ）和学生班级编号（ classNo ）进行分区。

```
create table if not exists student (
id int,
name STRING,
facultyNo int,
classNo INT)
using csv
options (path 'path 'obs://bucketName/filePath'')
partitioned by (faculytNo, classNo);
```

步骤2 在表格中插入分区数据。

利用插入数据中的内容，可以插入以下数据

```
INSERT into student
partition (facultyNo = 10, classNo = 101)
values (1010101, "student01"), (1010102, "student02");

INSERT into student
partition (facultyNo = 10, classNo = 102)
values (1010203, "student03"), (1010204, "student04");
```



```
INSERT into student
partition (facultyNo = 20, classNo = 101)
values (2010105, "student05"), (2010106, "student06");

INSERT into student
partition (facultyNo = 20, classNo = 102)
values (2010207, "student07"), (2010208, "student08");

INSERT into student
partition (facultyNo = 20, classNo = 103)
values (2010309, "student09"), (2010310, "student10");

INSERT into student
partition (facultyNo = 30, classNo = 101)
values (3010111, "student11"), (3010112, "student12");

INSERT into student
partition (facultyNo = 30, classNo = 102)
values (3010213, "student13"), (3010214, "student14");
```

步骤3 查看分区。

利用查看指定表所有分区中的内容，可以查看相关的分区内容。

示例代码如下：

```
SHOW partitions student;
```

表 1-30 表数据示例

facultyNo	classNo
facultyNo=10	classNo=101
facultyNo=10	classNo=102
facultyNo=20	classNo=101
facultyNo=20	classNo=102
facultyNo=20	classNo=103
facultyNo=30	classNo=101
facultyNo=30	classNo=102

步骤4 删除分区。

说明

本节操作介绍按指定筛选条件删除分区。无需指定筛选条件请参考**删除分区**。

本示例与**删除分区**不可混合使用。请区分此处关键字“partitions”与**删除分区**的关键字“partition”。

- **示例1：按指定筛选条件删除分区（该操作仅适用于OBS表），使用AND语句删除分区数据。**

表 1-31 执行前数据

facultyNo	classNo
facultyNo=10	classNo=101
facultyNo=10	classNo=102
facultyNo=20	classNo=101
facultyNo=20	classNo=102

执行以下语句删除facultyNo = 20且classNo = 102分区：

```
ALTER TABLE student
DROP IF EXISTS
PARTITIONS (facultyNo = 20 AND classNo = 102);
```

可以看到该语句会删除同时满足（AND）两个条件的分支的分区。

表 1-32 执行后数据

facultyNo	classNo
facultyNo=10	classNo=101
facultyNo=10	classNo=102
facultyNo=20	classNo=101

- **示例2：按指定筛选条件删除分区（该操作仅适用于OBS表），使用OR语句进行删除。**

表 1-33 执行前数据

facultyNo	classNo
facultyNo=10	classNo=101
facultyNo=10	classNo=102
facultyNo=20	classNo=101
facultyNo=20	classNo=102

执行语句删除满足条件facultyNo = 10或classNo = 101的分区：

```
ALTER TABLE student
DROP IF EXISTS
PARTITIONS (facultyNo = 10),
PARTITIONS (classNo = 101);
```

执行结果：

表 1-34 执行后数据

facultyNo	classNo
facultyNo=20	classNo=102

在上述删除条件的框选下，分区记录中第一条数据既满足院系编号，又满足班级编号，第二条数据满足了院系编号，第三条数据满足了班级编号。

因此执行删除分区语句后只剩余1行分区。

按照方法一，上述执行语句还可以写成：

```
ALTER TABLE student
DROP IF EXISTS
PARTITIONS (facultyNo = 10 OR classNo = 101);
```

- **示例3：按指定筛选条件删除分区（该操作仅适用于OBS表），使用关系运算符语句删除指定分区。**

表 1-35 执行前数据

facultyNo	classNo
facultyNo=10	classNo=101
facultyNo=10	classNo=102
facultyNo=20	classNo=101
facultyNo=20	classNo=102
facultyNo=20	classNo=103

执行删除分区语句，删除classNo大于100小于102的分区：

```
ALTER TABLE student
DROP IF EXISTS
PARTITIONS (classNo BETWEEN 100 AND 102);
```

执行结果：

表 1-36 执行前数据

facultyNo	classNo
facultyNo=20	classNo=103

----结束

1.10.5 修改表分区位置（只支持 OBS 表）

功能描述

修改表分区的位置。

语法格式

```
ALTER TABLE table_name
  PARTITION partition_specs
  SET LOCATION obs_path;
```

关键字

- PARTITION：分区。
- LOCATION：分区路径。

参数说明

表 1-37 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。
obs_path	OBS存储路径。

注意事项

- 所要修改位置的表分区必须是已经存在的，否则将报错。
- “partition_specs”中的参数默认带有“( )”，例如：**PARTITION (dt='2009-09-09',city='xxx')**。
- 所指定的新的OBS路径必须是已经存在的绝对路径，否则将报错。
- 若新增分区指定的路径包含子目录（或嵌套子目录），则子目录下面的所有文件类型及内容也将作为该分区的记录。用户需要保证该分区目录下所有文件类型和文件内容与表的字段一致，否则查询将报错。

示例

将student表的分区dt='2008-08-08',city='xxx'的OBS路径设置为“obs://bucketName/fileName/student/dt=2008-08-08/city=xxx”。

```
ALTER TABLE student
  PARTITION(dt='2008-08-08',city='xxx')
  SET LOCATION 'obs://bucketName/fileName/student/dt=2008-08-08/city=xxx';
```

1.10.6 更新表分区信息（只支持 OBS 表）

功能描述

更新表在元数据库中的分区信息。

语法格式

```
MSCK REPAIR TABLE table_name;
```

或


```
ALTER TABLE table_name RECOVER PARTITIONS;
```

关键字

- PARTITIONS：分区。
- SERDEPROPERTIES：Serde属性。

参数说明

表 1-38 参数描述

参数	描述
table_name	表名称。
partition_specs	分区字段。
obs_path	OBS存储路径。

注意事项

- 该命令的主要应用场景是针对分区表，如当手动在OBS上面添加分区目录时，再通过上述命令将该新增的分区信息刷新到元数据库中，通过“SHOW PARTITIONS table_name”命令查看新增的分区。
- 分区目录名称必须按照指定的格式输入，即“tablepath/partition_column_name=partition_column_value”。

示例

下述两语句都将更新表ptable在元数据库中的分区信息。

```
MSCK REPAIR TABLE ptable;
```

或

```
ALTER TABLE ptable RECOVER PARTITIONS;
```

1.10.7 REFRESH TABLE 刷新表元数据

功能描述

Spark为了提高性能会缓存Parquet的元数据信息。当更新了Parquet表时，缓存的元数据信息未更新，导致Spark SQL查询不到新插入的数据作业执行报错，报错信息参考如下：

```
DLI.0002: FileNotFoundException: getFileStatus on  error message
```

该场景下就需要使用REFRESH TABLE来解决该问题。REFRESH TABLE是用于重新整理某个分区的文件，重用之前的表元数据信息，能够检测到表的字段的增加或者减少，主要用于表中元数据未修改，表的数据修改的场景。

语法格式

```
REFRESH TABLE [db_name.]table_name;
```

关键字

无。

参数说明

表 1-39 参数描述

参数	描述
db_name	Database名称，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。
table_name	表名称。Database中的表名，由字母、数字和下划线（_）组成。不能是纯数字，且不能以下划线开头。匹配规则为：^(?!_)(?![0-9]+$)[A-Za-z0-9_$]*$。如果特殊字符需要使用单引号（''）包围起来。

注意事项

无。

示例

刷新表test的元数据信息。

```
REFRESH TABLE test;
```

1.11 导入数据

功能描述

LOAD DATA可用于导入CSV、Parquet、ORC、JSON、Avro格式的数据，内部将转换成Parquet数据格式进行存储。

语法格式

```
LOAD DATA INPATH 'folder_path' INTO TABLE [db_name.]table_name
  OPTIONS(property_name=property_value, ...);
```

关键字

- INPATH：数据路径。
- OPTIONS：属性列表。

参数说明

表 1-40 参数描述

参数	描述
folder_path	原始数据文件夹或者文件的OBS路径。
db_name	数据库名称。若未指定，则使用当前数据库。
table_name	需要导入数据的表的名称。

以下是可以在导入数据时使用的配置选项：

- DATA_TYPE: 指定导入的数据类型，当前支持CSV、Parquet、ORC、JSON、Avro类型，默认值为“CSV”。

  配置项为OPTIONS('DATA_TYPE'='CSV')

  导入CSV和JSON文件时，有三种模式可以选择：

  - PERMISSIVE：选择PERMISSIVE模式时，如果某一列数据类型与目标表列数据类型不匹配，则该行数据将被设置为null。
  - DROPMALFORMED：选择DROPMALFORMED模式时，如果某一列数据类型与目标表列数据类型不匹配，则不导入该行数据。
  - FAILFAST：选择FAILFAST模式时，如果某一列类型不匹配，则会抛出异常，导入失败。

  模式设置可通过在OPTIONS中添加 OPTIONS('MODE'='PERMISSIVE')进行设置。
- DELIMITER：可以在导入命令中指定分隔符，默认值为“,”。

  配置项为OPTIONS('DELIMITER'=',')。

  对于CSV数据，支持如下所述分隔符：

  - 制表符tab，例如：'DELIMITER'='\t'。
  - 任意的二进制字符，例如：'DELIMITER'='\u0001(^A)'。
  - 单引号（'），单引号必须在双引号（" "）内。例如：'DELIMITER'= "'"。
- QUOTECHAR：可以在导入命令中指定引号字符。默认值为"。

  配置项为OPTIONS('QUOTECHAR'='"')
- COMMENTCHAR：可以在导入命令中指定注释字符。在导入操作期间，如果在行的开头遇到注释字符，那么该行将被视为注释，并且不会被导入。默认值为 # 。

  配置项为*OPTIONS('COMMENTCHAR'='#')*
- HEADER：用来表示源文件是否有表头。取值范围为“true”和“false”。“true”表示有表头，“false”表示无表头。默认值为“false”。如果没有表头，可以在导入命令中指定FILEHEADER参数提供表头。

  配置项为*OPTIONS('HEADER'='true')*
- FILEHEADER：如果源文件中没有表头，可在LOAD DATA命令中提供表头。

  OPTIONS('FILEHEADER'='column1,column2')
- ESCAPECHAR：如果用户想在CSV上对Escape字符进行严格验证，可以提供Escape字符。默认值为“\\”。

  配置项为OPTIONS('ESCAPECHAR'='\\')

说明

如果在CSV数据中输入ESCAPECHAR，该ESCAPECHAR必须在双引号（ " " ）内。例如："a \b"。

- MAXCOLUMNS：该可选参数指定了在一行中，CSV解析器解析的最大列数。
  配置项为*OPTIONS('MAXCOLUMNS'='400')*

表 1-41 MAXCOLUMNS

可选参数名称	默认值	最大值
MAXCOLUMNS	2000	20000

说明

设置MAXCOLUMNS Option的值后，导入数据会对executor的内存有要求，所以导入数据可能会由于executor内存不足而失败。

- DATEFORMAT：指定列的日期格式。
  OPTIONS('DATEFORMAT'='dateFormat')

说明

- 默认值为：yyyy-MM-dd。
- 日期格式由Java的日期模式字符串指定。在Java的日期和时间模式字符串中，未加单引号(')的字符'A' 到'Z' 和'a' 到'z' 被解释为模式字符，用来表示日期或时间字符串元素。若模式字符使用单引号 (') 引起来，则在解析时只进行文本匹配，而不进行解析。Java模式字符定义请参见**表1-42**。

表 1-42 日期及时间模式字符定义

模式字符	日期或时间元素	示例
G	纪元标识符	AD
y	年份	1996; 96
M	月份	July; Jul; 07
w	年中的周数	27(该年的第27周)
W	月中的周数	2(该月的第2周)
D	年中的天数	189(该年的第189天)
d	月中的天数	10(该月的第10天)
u	星期中的天数	1 = 星期一, ..., 7 = 星期日
a	am/pm 标记	pm(下午时)
H	24小时数(0-23)	2
h	12小时数(1-12)	12

模式字符	日期或时间元素	示例
m	分钟数	30
s	秒数	55
S	毫秒数	978
z	时区	Pacific Standard Time; PST; GMT-08:00

- TIMESTAMPFORMAT：指定列的时间戳格式。

  OPTIONS('TIMESTAMPFORMAT'='timestampFormat')

  说明

  - 默认值为：yyyy-MM-dd HH:mm:ss。
  - 时间戳格式由Java的时间模式字符串指定。Java时间模式字符串定义详见**表3 日期及时间模式字符定义**。

- MODE：指定导入过程错误记录的处理模式，支持三种选项：PERMISSIVE、DROPMALFORMED和FAILFAST。

  OPTIONS('MODE'='permissive')

  说明

  - PERMISSIVE（默认）：尽可能地解析bad records，如果遇到不能转换的字段，则整行为null
  - DROPMALFORMED：忽略掉无法解析的bad records
  - FAILFAST：遇到无法解析的记录时，抛出异常并使Job失败

- BADRECORDSPATH：指定导入过程中错误记录的存储目录。

  OPTIONS('BADRECORDSPATH'='obs://bucket/path')

  说明

  配置该选项后，MODE不可配，固定为"DROPMALFORMED"，即将能够成功转换的记录导入到目标表，而将转换失败的记录存储到指定错误记录存储目录。

注意事项

- 导入OBS表时，创建OBS表时指定的路径必须是文件夹，若建表路径是文件将导致导入数据失败。
- 仅支持导入位于OBS路径上的原始数据。
- 不建议对同一张表并发导入数据，因为有一定概率发生并发冲突，导致导入失败。
- 导入数据时只能指定一个路径，路径中不能包含逗号。
- 当OBS桶目录下有文件夹和文件同名时，导入数据会优先指向该路径下的文件而非文件夹。
- 导入PARQUET、ORC及JSON类型数据时，必须指定*DATA_TYPE*这一OPTIONS，否则会以默认的“CSV”格式进行解析，从而导致导入的数据格式不正确。
- 导入CSV及JSON类型数据时，如果包含日期及时间列，需要指定*DATEFORMAT*及*TIMESTAMPFORMAT*选项，否则将以默认的日期及时间戳格式进行解析。

示例

说明

导入数据前已参考**创建OBS表**中的示例描述创建对应的表。

- 可使用下列语句将CSV文件导入到表，“t”为表名。

```
LOAD DATA INPATH 'obs://dli/data.csv' INTO TABLE t
  OPTIONS('DELIMITER'=',' , 'QUOTECHAR'='"','COMMENTCHAR'='#','HEADER'='false');
```

- 可使用下列语句将JSON文件导入到表，“jsontb”为表名。

```
LOAD DATA INPATH 'obs://dli/alltype.json' into table jsontb
  OPTIONS('DATA_TYPE'='json','DATEFORMAT'='yyyy/MM/dd','TIMESTAMPFORMAT'='yyyy/MM/dd
HH:mm:ss');
```

1.12 插入数据

功能描述

将SELECT查询结果或某条数据插入到表中。

约束限制

- insert overwrite语法不适用于“自读自写”场景，该场景因涉及数据的连续处理和更新，如果使用insert overwrite语法可能存在数据丢失风险。

  "自读自写"是指在处理数据时能够读取数据，同时根据读取的数据生成新的数据或对数据进行修改。

- 使用Hive和Datasource（除Hudi外）表在执行数据修改类命令（例如insert into，load data）时由于数据源不支持事务性，在系统故障或队列资源重启后，可能会导致数据重复或数据不一致等问题。

  为了避免这种情况，建议优先选择支持事务性的数据源，如Hudi类型数据源，该类数据源具备ACID（Atomicity、Consistency、Isolation、Durability）能力，有助于确保数据的一致性和准确性。

  了解更多：**执行Insert into后数据重复怎么办?**

语法格式

- 将SELECT查询结果插入到表中

```
INSERT INTO [TABLE] [db_name.]table_name
  [PARTITION part_spec] select_statement;
INSERT OVERWRITE TABLE [db_name.]table_name
  [PARTITION part_spec] select_statement;
part_spec:
  : (part_col_name1=val1 [, part_col_name2=val2, ...])
```

- 将某条数据插入到表中

```
INSERT INTO [TABLE] [db_name.]table_name
  [PARTITION part_spec] VALUES values_row [, values_row ...];
INSERT OVERWRITE TABLE [db_name.]table_name
  [PARTITION part_spec] VALUES values_row [, values_row ...];
values_row:
  : (val1 [, val2, ...])
```

关键字

表 1-43 INSERT 关键字说明

参数	描述
db_name	需要执行INSERT命令的表所在数据库的名称。
table_name	需要执行INSERT命令的表的名称。
part_spec	指定详细的分区信息。若分区字段为多个字段，需要包含所有的字段，但是可以不包含对应的值，系统会匹配上对应的分区。单表分区数最多允许100000个。
select_statement	源表上的SELECT查询。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

- 表必须已经存在。
- 如果动态分区不需要指定分区，则将“part_spec”作为普通字段放置SELECT语句中。
- 被插入的OBS表在建表时只能指定文件夹路径。
- 源表和目标表的数据类型和列字段个数应该相同，否则插入失败。
- 不建议对同一张表并发插入数据，可能会由于并发冲突导致插入数据结果异常。
- INSERT INTO命令用于将查询的结果追加到目标表中。
- INSERT OVERWRITE命令用于覆盖源表中已有的数据。
- INSERT INTO命令可以并行执行，INSERT OVERWRITE命令只有在分区表下不同的插入到不同静态分区才可以并行。
- INSERT INTO命令和INSERT OVERWRITE命令同时执行，其结果是未知的。
- 在从源表插入数据到目标表的过程中，无法在源表中导入或更新数据。
- 对于Hive分区表的动态INSERT OVERWRITE，支持覆盖涉及到的分区数据，不支持覆盖整表数据。
- 如果需要覆盖DataSource表指定分区数据，需要先配置参数：dli.sql.dynamicPartitionOverwrite.enabled=true，再通过“insert overwrite”语句实现，“dli.sql.dynamicPartitionOverwrite.enabled”默认值为“false”，表示覆盖整表数据。例如：

  ```
  insert overwrite table tb1 partition(part1='v1', part2='v2') select * from ...
  ```

  说明

  在“数据湖探索管理控制台>SQL编辑器”页面，单击编辑窗口右上角“设置”，可配置参数。

- 通过配置“spark.sql.shuffle.partitions”参数可以设置表在OBS桶中插入的文件个数，同时，为了避免数据倾斜，在INSERT语句后可加上“distribute by rand()”，可以增加处理作业的并发量。例如：

  ```
  insert into table table_target select * from table_source distribute by cast(rand() * N as int);
  ```


示例

说明

导入数据前已参考**创建OBS表**中的示例描述创建对应的表。

- 将SELECT查询结果插入到表中
  - 使用DataSource语法创建一个parquet格式的分区表
    ```
    CREATE TABLE data_source_tab1 (col1 INT, p1 INT, p2 INT)
     USING PARQUET PARTITIONED BY (p1, p2);
    ```
  - 插入查询结果到分区 (p1 = 3, p2 = 4)中
    ```
    INSERT INTO data_source_tab1 PARTITION (p1 = 3, p2 = 4)
     SELECT id FROM RANGE(1, 3);
    ```
  - 插入新的查询结果到分区 (p1 = 3, p2 = 4) 中
    ```
    INSERT OVERWRITE TABLE data_source_tab1 PARTITION (p1 = 3, p2 = 4)
     SELECT id FROM RANGE(3, 5);
    ```
- 将某条数据插入表中
  - 使用Hive语法创建一个parquet格式的分区表
    ```
    CREATE TABLE hive_serde_tab1 (col1 INT, p1 INT, p2 INT)
     USING HIVE OPTIONS(fileFormat 'PARQUET') PARTITIONED BY (p1, p2);
    ```
  - 插入两条数据到分区 (p1 = 3, p2 = 4)中
    ```
    INSERT INTO hive_serde_tab1 PARTITION (p1 = 3, p2 = 4)
     VALUES (1), (2);
    ```
  - 插入新的数据到分区 (p1 = 3, p2 = 4) 中
    ```
    INSERT OVERWRITE TABLE hive_serde_tab1 PARTITION (p1 = 3, p2 = 4)
     VALUES (3), (4);
    ```

执行 Insert into 后数据重复怎么办?

- **问题现象:**

  使用Hive和Datasource（除Hudi外）表在执行数据修改类命令（例如insert into，load data）时由于数据源不支持事务性，在系统故障或队列资源重启后，可能会导致数据重复或数据不一致等问题。

- **原因分析:**

  在数据的Commit阶段如果出现队列资源重启可能会导致数据已经被修复到正式目录中。如果执行的是Insert into语句，资源重启后触发重试就会有概率导致数据重复写入。

- **解决方案:**

  a. 推荐使用具备ACID能力的Hudi类型数据源。

  b. 建议尽量使用insert overwrite这样幂等的语法而不是insert into等非幂等语法插入数据。

  c. 如果严格需求数据不能重复，建议在insert into后对表数据执行去重操作，防止数据重复。

1.13 清空数据

功能描述

清除表的数据。

语法格式

```
TRUNCATE TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)];
```

关键字

表 1-44 关键字说明

参数	描述
tablename	需要执行Truncate命令的表的名称。
partcol1	需要删除的表的分区名称。

注意事项

只支持清除表的数据。

示例

```
truncate table test PARTITION (class = 'test');
```

1.14 导出查询结果

功能描述

INSERT OVERWRITE DIRECTORY用于将查询结果直接写入到指定的目录，支持按CSV、Parquet、ORC、JSON、Avro格式进行存储。

语法格式

```
INSERT OVERWRITE DIRECTORY path
  USING file_format
  [OPTIONS(key1=value1)]
  select_statement;
```

关键字

- USING：指定所存储格式。
- OPTIONS：导出时的属性列表，为可选项。

参数

表 1-45 INSERT OVERWRITE DIRECTORY 参数描述

参数	描述
path	要将查询结果写入的OBS路径。
file_format	写入的文件格式，支持按CSV、Parquet、ORC、JSON、Avro格式。

说明

file_format为csv时，options参数可以参考**表1-10**。

注意事项

- 通过配置“spark.sql.shuffle.partitions”参数可以设置表在OBS桶中插入的文件个数，同时，为了避免数据倾斜，在INSERT语句后可加上“distribute by rand()”，可以增加处理作业的并发量。例如：
  ```
  insert into table table_target select * from table_source distribute by cast(rand() * N as int);
  ```
- 配置项为OPTIONS('DELIMITER'=',')时，可以指定分隔符，默认值为“,”。

  对于CSV数据，支持如下所述分隔符：
  - 制表符tab，例如：'DELIMITER'='\t'。
  - 支持通过unicode编码指定分隔符，例如：'DELIMITER'='\u0001'。
  - 单引号（'），单引号必须在双引号（" "）内。例如：'DELIMITER'= "'"。

示例

```
INSERT OVERWRITE DIRECTORY 'obs://bucket/dir'
  USING csv
  OPTIONS(key1=value1)
  select * from db1.tb1;
```

1.15 表生命周期管理

1.15.1 创建表时指定表的生命周期

功能描述

DLI提供了表生命周期管理功能，在创建表时指定表的生命周期。DLI会根据每张表的最后修改时间和表的生命周期来判断是否要回收此表。通过设置表的生命周期，可以帮助您更好的管理数目众多的表，自动清理长期不再使用的数据表，简化数据表的回收流程。同时支持数据恢复设置，避免因误操作丢失数据。

表的回收规则

- 在创建表时通过TBLPROPERTIES指定表的生命周期。
  - **非分区表**

    如果表是非分区表，根据每张表的最后修改时间，经过生命周期时间后判断是否要回收此表。
  - **分区表**

    如果是分区表，则根据各分区的最后一次表数据被修改的时间（LAST_ACCESS_TIME）判断该分区是否该被回收。分区表的最后一个分区被回收后，该表不会被删除。

    分区表不支持设置分区级的生命周期，仅支持表级别的生命周期管理。
- 生命周期回收为每天定时启动，扫描全量分区。

  生命周期回收为每天定时启动，扫描全量分区的最后一次表数据被修改的时间（LAST_ACCESS_TIME）需要超过生命周期指定的时间才回收。

假设某个分区表生命周期为1天，该分区数据最后一次被修改的时间是2023年05月20日15时。如果在2023年05月20日15时之前扫描此表（不到一天），则不会回收表分区。如果2023年05月20日回收扫描时发现表分区最后一次表数据被修改的时间（LAST_ACCESS_TIME）超过生命周期指定的时间，则上述分区会被回收。

- 生命周期主要提供定期回收表或分区的功能，每天根据服务的繁忙程度，不定时回收。不能确保表或分区的生命周期到期后，立刻被回收。
- 删除表后，表的所有属性信息全部会删除，包括生命周期。新建同名表后，表的生命周期以新设置的属性为准。

约束限制

- 使用生命周期前需要在“全局配置 > 服务授权 > 委托权限设置”中，对（Tenant Administrator(项目级)）授权。
- 表生命周期功能支持Hive、DataSource语法创建表、多版本表，暂不支持跨源表、Carbon表。
- 生命周期单位为天，取值为正整数。
- 生命周期只能在表级别设置，不能在分区级设置。为分区表指定的生命周期，适用于该表所有的分区。
- 生命周期设置后，DLI表和OBS表支持数据备份，OBS表的备份目录需要手工设置。且备份目录应选择在并行文件系统上，备份目录必须和原表目录在同一个桶上，备份目录不能与原表相同目录或者子目录同名。

语法格式

- **DataSource语法创建DLI表**

```
CREATE TABLE table_name(name string, id int)
USING parquet
TBLPROPERTIES( "dli.lifecycle.days"=1 );
```

- **Hive语法创建DLI表**

```
CREATE TABLE table_name(name string, id int)
stored as parquet
TBLPROPERTIES( "dli.lifecycle.days"=1 );
```

- **DataSource语法创建OBS表**

```
CREATE TABLE table_name(name string, id int)
USING parquet
OPTIONS (path "obs://dli-test/table_name")
TBLPROPERTIES( "dli.lifecycle.days"=1, "external.table.purge"='true', "dli.lifecycle.trash.dir"='obs://dli-test/Lifecycle-Trash' );
```

- **Hive语法创建OBS表**

```
CREATE TABLE table_name(name string, id int)
STORED AS parquet
LOCATION 'obs://dli-test/table_name'
TBLPROPERTIES( "dli.lifecycle.days"=1, "external.table.purge"='true', "dli.lifecycle.trash.dir"='obs://dli-test/Lifecycle-Trash' );
```

关键字

- TBLPROPERTIES：表的属性，增加表的生命周期功能。
- OPTIONS：新建表的路径，适用于DataSource语法创建OBS表。
- LOCATION:新建表的路径，适用于Hive语法创建OBS表。

参数说明

表 1-46 参数说明

参数名称	是否必选	参数说明
table_name	是	需要设置生命周期的表名。
dli.lifecycle.days	是	设置的生命周期时间，只能为正整数，单位为天。
external.table.purge	否	仅OBS表支持配置该参数。 是否需要在删除表或分区时，清除path路径下的数据。默认不删除。 设置'external.table.purge'='true'时： ● 非分区OBS表配置删除文件后，表目录也会删除。 ● 分区OBS表自定义分区数据也会删除。
dli.lifecycle.trash.dir	否	仅OBS表支持配置该参数。 设置'external.table.purge'='true'时，清除数据的备份目录，默认七天后删除备份数据。

示例

- **DataSource语法新建test_datasource_lifecycle表，生命周期为100天**

```
CREATE TABLE test_datasource_lifecycle(id int)
USING parquet
TBLPROPERTIES( "dli.lifecycle.days"=100);
```

- **Hive语法新建test_hive_lifecycle表，生命周期为100天。**

```
CREATE TABLE test_hive_lifecycle(id int)
stored as parquet
TBLPROPERTIES( "dli.lifecycle.days"=100);
```

- **DataSource语法新建test_datasource_lifecycle_obs表，生命周期为100天，过期时默认删除数据且数据备份至目录'obs://dli-test/'**。

```
CREATE TABLE test_datasource_lifecycle_obs(name string, id int)
USING parquet
OPTIONS (path "obs://dli-test/xxx")
TBLPROPERTIES( "dli.lifecycle.days"=100, "external.table.purge"='true', "dli.lifecycle.trash.dir"='obs://dli-test/Lifecycle-Trash' );
```

- **Hive语法新建test_hive_lifecycle_obs表，生命周期为100天，过期时默认删除数据且数据备份至目录'obs://dli-test/'**。

```
CREATE TABLE test_hive_lifecycle_obs(name string, id int)
STORED AS parquet
LOCATION 'obs://dli-test/xxx'
TBLPROPERTIES( "dli.lifecycle.days"=100, "external.table.purge"='true', "dli.lifecycle.trash.dir"='obs://dli-test/Lifecycle-Trash' );
```

1.15.2 修改表生命周期的时间

功能描述

修改已存在的分区表或非分区表的生命周期。

当第一次开启生命周期时，会扫描表/分区会扫描路径下的表数据文件，更新表/分区的LAST_ACCESS_TIME，耗时与分区数和文件数相关。

约束限制

- 表生命周期功能支持Hive、DataSource语法创建表、多版本表，暂不支持跨源表、Carbon表。
- 生命周期单位为天，取值为正整数。
- 生命周期只能在表级别设置，不能在分区级设置。为分区表指定的生命周期，适用于该表所有的分区。

语法格式

```
ALTER TABLE table_name
SET TBLPROPERTIES("dli.lifecycle.days"='N')
```

关键字

TBLPROPERTIES：表的属性增加表的生命周期功能。

参数说明

表 1-47 修改表的生命周期参数说明

参数名称	是否必选	参数说明
table_name	是	需要修改生命周期的表名。
dli.lifecycle.days	是	修改后的生命周期时间，只能为正整数，单位为天。

示例

- 示例1：修改表的生命周期，开启test_lifecycle_exists表生命周期，并将生命周期设为50天。
  ```
  alter table test_lifecycle_exists
  SET TBLPROPERTIES("dli.lifecycle.days"='50');
  ```
- 示例2：对已存在且未设置生命周期的分区表或非分区表开启表的生命周期,开启test_lifecycle_exists表生命周期，并将生命周期设为50天。
  ```
  alter table test_lifecycle_exists
  SET TBLPROPERTIES(
      "dli.lifecycle.days"='50',
      "dli.table.lifecycle.status"='enable'
  );
  ```

1.15.3 禁止或恢复表的生命周期

功能介绍

禁止或恢复指定表或分区的生命周期。

使用禁止或恢复表的生命周期有以下两种场景：

1. 表或分区表开启了生命周期的功能，该功能可以禁止或恢复表的生命周期，即修改“dli.table.lifecycle.status”的参数值。
2. 表或分区表未开启生命周期的功能，使用禁止或恢复表的生命周期，则会增加“dli.table.lifecycle.status”这一属性。

约束限制

- 表生命周期功能支持Hive、DataSource语法创建表、多版本表，暂不支持跨源表、Carbon表。
- 生命周期单位为天，取值为正整数。
- 生命周期只能在表级别设置，不能在分区级设置。为分区表指定的生命周期，适用于该表所有的分区。

语法格式

- 该语法在表级别禁止或恢复表的生命周期
  ```
  ALTER TABLE table_name SET TBLPROPERTIES("dli.table.lifecycle.status"={enable|disable});
  ```
- 该语法可在表或分区表级别禁止或恢复指定表
  ```
  ALTER TABLE table_name [pt_spec] LIFECYCLE {enable|disable};
  ```

关键字

TBLPROPERTIES：表的属性，增加表的生命周期功能。

参数说明

表 1-48 禁止或恢复生命周期参数说明

参数名称	是否必选	参数说明
table_name	是	待禁止或恢复生命周期的表的名称。
pt_spec	否	待禁止或恢复生命周期的表的分区信息。格式为partition_col1=col1_value1, partition_col2=col2_value1...。对于有多级分区的表，必须指明全部的分区值。
enable	否	恢复表或指定分区的生命周期功能 • 表及其分区重新参与生命周期回收，默认使用当前表及分区上的生命周期配置。 • 开启表生命周期前可以修改表及分区的生命周期配置，防止开启表生命周期后因使用之前的配置导致数据被误回收。

参数名称	是否必选	参数说明
disable	否	禁止表或指定分区的生命周期功能。 • 禁止表本身及其所有分区被生命周期回收，优先级高于恢复表分区生命周期。即当使用禁止表或指定分区的生命周期功能时，设置待禁止或恢复生命周期的表的分区信息是无效的。 • 禁止表的生命周期功能后，表的生命周期配置及其分区的enable和disable标记会被保留。 • 禁止表的生命周期功能后，仍然可以修改表及分区表的生命周期配置。

示例

- 示例1：禁止表test_lifecycle的生命周期功能。
```
alter table test_lifecycle SET TBLPROPERTIES("dli.table.lifecycle.status"='disable');
```
- 示例2：禁止表test_lifecycle中时间为20230520分区的生命周期功能。
```
alter table test_lifecycle partition (dt='20230520') LIFECYCLE 'disable';
```

说明

- 当设置禁止分区表的生命周期功能后，该表的所有分区的生命周期功能都会被禁止。

1.16 跨源连接 HBase 表

1.16.1 创建表关联 HBase

功能描述

使用CREATE TABLE命令创建表并关联HBase上已有的表。

说明

Spark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提供的跨源认证方式。

前提条件

- 创建表关联HBase之前需要创建跨源连接。
- 请确保在DLI队列host文件中添加MRS集群master节点的“/etc/hosts”信息。
- 该语法不支持安全集群。

语法格式

- 单个RowKey
```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME (
  ATTR1 TYPE,
  ATTR2 TYPE,
  ATTR3 TYPE)
  USING [HBASE] OPTIONS (
```

```
'ZKHost'='xx',
'TableName'='TABLE_IN_HBASE',
'RowKey'='ATTR1',
'Cols'='ATTR2:CF1.C1, ATTR3:CF1.C2');
```

- 组合RowKey

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME (
  ATTR1 String,
  ATTR2 String,
  ATTR3 TYPE)
  USING [HBASE] OPTIONS (
  'ZKHost'='xx',
  'TableName'='TABLE_IN_HBASE',
  'RowKey'='ATTR1:2, ATTR2:10',
  'Cols'='ATTR2:CF1.C1, ATTR3:CF1.C2'
```

关键字

表 1-49 CREATE TABLE 关键字说明

参数	描述
USING [HBASE]	指定hbase datasource，大小写不敏感。
ZKHost	HBase集群的ZK连接地址。 获取ZK连接地址需要先创建跨源连接。 ● 访问MRS集群，填写ZK所在节点IP与ZK对外端口，格式为："ZK_IP1:ZK_PORT1,ZK_IP2:ZK_PORT2"。 **说明**
TableName	指定在HBase集群中已创建的表名。
RowKey	指定作为rowkey的dli关联表字段，支持单rowkey与组合rowkey。单rowkey支持数值与String类型，不需要指定长度。组合rowkey仅支持String类型定长数据，格式为：属性名1:长度,属性名2:长度。
Cols	通过逗号分隔的表字段与HBase表的列之间的对应关系。其中，冒号前面放置表字段，冒号后面放置HBase表信息，用‘.’分隔HBase表的列族与列名。

注意事项

- 若所要创建的表已经存在将报错，可以通过添加IF NOT EXISTS参数跳过该错误。
- OPTIONS中的所有参数是必选的，参数名称大小写不敏感，但参数值大小写敏感。
- OPTIONS中引号内的值前后不能带空格，空格也会被当做有效值。
- 表名及列名的描述仅支持字符串常量。
- 创建表时要说明列名及对应的数据类型，目前支持的数据类型为：boolean、short、int、long、float、double和string。
- 作为RowKey的字段（如上述语法格式中的ATTR1），其值不能为null，长度要大于0，小于或等于32767。

- Cols与RowKey中的字段加起来的数量必须与表的字段保持一致，即表中所有的字段都到对应到Cols和RowKey中，但是顺序可以任意。
- 组合Rowkey只支持String类型，在使用组合Rowkey时，每个属性后面必须带上长度。当Rowkey指定的字段只有一个的时候，该字段的类型可以是支持的所有数据类型，并且不需要填写长度。
- 在组合Rowkey的场景中
  - 插入Rowkey数据时，如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要短，则会在数据后面补'\0'字符；如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要长，则会在实际插入HBase的时候进行截断。
  - 读取HBase上的Rowkey数据时，如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要短，则会抛出异常（OutofBoundException）；如果某个属性的实际数据的长度比属性作为Rowkey时指定的长度要长，则会在读取时进行截断。

示例

```
CREATE TABLE test_hbase(
ATTR1 int,
ATTR2 int,
ATTR3 string)
using hbase OPTIONS (
'ZKHost'='to-hbase-1174405101-CE1bDm5B.datasource.com:2181',
'TableName'='HBASE_TABLE',
'RowKey'='ATTR1',
'Cols'='ATTR2:CF1.C1, ATTR3:CF1.C2');
```

1.16.2 插入数据至 HBase 表

功能描述

INSERT INTO命令将表中的数据插入到已关联的hbase表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**SELECT基本语句**。

参数说明

表 1-50 参数描述

参数	描述
DLI_TABLE	已创建跨源连接的表名称。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

- 表必须已经存在。
- 在“**创建表关联HBase**”章节创建的表中，OPTIONS里的Cols指定的列族如果不存在，insert into执行时会报错。
- 如果插入的(rowkey, 列族, 列)已存在，则执行插入操作时，会覆盖hbase中相同的(rowkey, 列族, 列)。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。
- 不支持INSERT OVERWRITE语法。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

1.16.3 查询 HBase 表

SELECT命令用于查询hbase表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表中的数据。

```
SELECT * FROM test_hbase limit 100;
```

查询下压

通过hbase进行数据过滤，即HBase Client将过滤条件传给HBase服务端进行处理，HBase服务端只返回用户需要的数据，提高了Spark SQL查询的速度。对于HBase不支持的过滤条件，例如组合Rowkey的查询，直接由Spark SQL进行。

- 支持查询下压的场景
  - 数据类型场景
    - Int
    - boolean
    - short
    - long
    - double
    - string

      说明

      float类型数据不支持查询下压。
  - 过滤条件场景
    - 过滤条件为>,<,>=,<=,=,!=,and,or

      例如：

      ```
      select * from tableName where (column1 >= value1 and column2<= value2) or column3 != value3
      ```
    - 过滤条件为like 和 not like，支持前缀，后缀和包含匹配

      例如：

      ```
      select * from tableName where column1 like "%value" or column2 like "value%" or column3 like "%value%"
      ```
    - 过滤条件为IsNotNull()

      例如：

      ```
      select * from tableName where IsNotNull(column)
      ```
    - 过滤条件为in ,not in

      例如：

      ```
      select * from tableName where column1 in (value1,value2,value3)  and column2 not in (value4,value5,value6)
      ```

- 过滤条件为between _ and _
  例如：
  ```
  select * from tableName where column1 between value1 and value2
  ```
- 组合rowkey中的子rowkey过滤
  例如，组合Rowkey为column1+column2+column3，进行子rowkey查询：
  ```
  select * from tableName where column1= value1
  ```

- 不支持查询下压的场景
  - 数据类型场景
    除上述支持的数据类型外，其余复杂数据类型不支持查询下压。
  - 过滤条件场景
    - length，count，max，min，join，groupby，orderby，limit和avg等
    - 过滤条件为列比较
      例如：
      ```
      select * from tableName where column1 > (column2+column3)
      ```

1.17 跨源连接 OpenTSDB 表

1.17.1 创建表关联 OpenTSDB

功能描述

使用CREATE TABLE命令创建表并关联OpenTSDB上已有的metric，该语法支持MRS服务的OpenTSDB。

前提条件

创建表关联OpenTSDB之前需要创建跨源连接。

语法格式

```
CREATE TABLE [IF NOT EXISTS] UQUERY_OPENTSDB_TABLE_NAME
  USING OPENTSDB OPTIONS (
  'host' = 'xx;xx',
  'metric' = 'METRIC_NAME',
  'tags' = 'TAG1,TAG2');
```

关键字

表 1-51 CREATE TABLE 关键字描述

参数	描述
host	OpenTSDB连接地址。 获取OpenTSDB连接地址需要先创建跨源连接。 ● 访问MRS OpenTSDB，若使用增强型跨源连接，填写OpenTSDB所在节点IP与端口，格式为"IP:PORT"，OpenTSDB存在多个节点时，用分号间隔。
metric	所创建的表对应的OpenTSDB中的指标名称。
tags	metric对应的标签，用于归类、过滤、快速检索等操作。可以是1个到8个，以“,”分隔，包括对应metric下所有tagk的值。

注意事项

创建表时，不需要指定timestamp和value字段，系统会根据指定的tags自动构建字段，包含以下字段，其中TAG1和TAG2由tags指定。

- TAG1 String
- TAG2 String
- timestamp Timestamp
- value double

示例

```
CREATE table opentsdb_table
  USING OPENTSDB OPTIONS (
  'host' = 'opentsdb-3xcl8dir15m58z3.com:4242',
  'metric' = 'city.temp',
  'tags' = 'city,location');
```

1.17.2 插入数据至 OpenTSDB 表

功能描述

使用INSERT INTO命令将表中的数据插入到已关联的OpenTSDB metric中。

说明

若OpenTSDB上不存在metric，插入数据时会在OpenTSDB上自动创建一个新的metric。

语法格式

```
INSERT INTO TABLE TABLE_NAME SELECT * FROM DLI_TABLE;
INSERT INTO TABLE TABLE_NAME VALUES(XXX);
```

关键字

表 1-52 INSERT INTO 关键字说明

参数	描述
TABLE_NAME	所关联的OpenTSDB表名。
DLI_TABLE	创建的表名称。

注意事项

- 插入的数据不能为null；插入的数据相同，会覆盖原数据；插入的数据只有value值不同，也会覆盖原数据。
- 不支持INSERT OVERWRITE语法。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。
- 时间戳格式只支持yyyy-MM-dd hh:mm:ss。

示例

```
INSERT INTO TABLE opentsdb_table VALUES('xxx','xxx','2018-05-03 00:00:00',21);
```

1.17.3 查询 OpenTSDB 表

SELECT命令用于查询OpenTSDB表中的数据。

说明

- 若OpenTSDB上不存在metric，查询对应的表会报错。
- 若OpenTSDB开了安全模式，则访问时，需要设置conf:dli.sql.mrs.opentsdb.ssl.enabled=true

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表opentsdb_table中的数据。

```
SELECT * FROM opentsdb_table limit 100;
```

1.18 跨源连接 DWS 表

1.18.1 创建表关联 DWS

功能描述

使用CREATE TABLE命令创建表并关联DWS上已有的表。

说明

Spark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提供的跨源认证方式。

前提条件

创建表关联DWS之前需要创建跨源连接。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME
  USING JDBC OPTIONS (
  'url'='xx',
  'dbtable'='db_name_in_DWS.table_name_in_DWS',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

关键字

表 1-53 CREATE TABLE 关键字说明

参数	描述
url	DWS的连接地址，需要先创建跨源连接。 创建增强型跨源连接后，可以使用DWS提供的"JDBC连接字符串（内网）"，或者内网地址和内网端口访问，格式为"协议头://内网IP:内网端口/数据库名"，例如："jdbc:postgresql://192.168.0.77:8000/postgres"。 **说明** DWS的连接地址格式为："协议头://访问地址:访问端口/数据库名" 例如： jdbc:postgresql://to-dws-1174405119-ihlUr78j.datasource.com:8000/postgres 如果想要访问DWS中自定义数据库，请在这个连接里将"postgres"修改为对应的数据库名字。
dbtable	指定在DWS关联的表名，或者"模式名.表名"，例如：public.table_name。
user	（已废弃）DWS的用户名。
password	（已废弃）DWS集群的用户密码。

参数	描述
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《 数据湖探索用户指南 》>。
encryption	使用跨源密码认证时配置为“true”。
partitionColumn	读取数据时，用于设置并发使用的数值型字段。 **说明** • “partitionColumn”、“lowerBound”、“upperBound”、“numPartitions”四个参数必须同时设置，不支持仅设置其中某一个或某几个。 • 为了提升并发读取的性能，建议使用自增列。
lowerBound	partitionColumn设置的字段数据最小值，该值包含在返回结果中。
upperBound	partitionColumn设置的字段数据最大值，该值不包含在返回结果中。
numPartitions	读取数据时并发数。 **说明** 实际读取数据时，会根据“lowerBound”与“upperBound”，平均分配给每个task，获取其中一部分的数据。例如： 'partitionColumn'='id', 'lowerBound'='0', 'upperBound'='100', 'numPartitions'='2' 表示在DLI中会起2个并发task，一个task执行id>=0 and id < 50，另一个task执行id >=50 and id < 100。
fetchsize	读取数据时，每一批次获取数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
batchsize	写入数据时，每一批次写入数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
truncate	执行overwrite时是否不删除原表，直接执行清空表操作，取值范围： • true • false 默认为“false”，即在执行overwrite操作时，先将原表删除再重新建表。
isolationLevel	事务隔离级别，取值范围： • NONE • READ_UNCOMMITTED • READ_COMMITTED • REPEATABLE_READ • SERIALIZABLE 默认值为“READ_UNCOMMITTED”。

注意事项

创建DWS关联表时，不需要指定关联表的Schema。DLI会自动获取DWS中对应参数"dbtable"中的表的Schema。

示例

```
CREATE TABLE IF NOT EXISTS dli_to_dws
  USING JDBC OPTIONS (
  'url'='jdbc:postgresql://to-dws-1174405119-ih1Ur78j.datasource.com:8000/postgres',
  'dbtable'='test_dws',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

1.18.2 插入数据至 DWS 表

功能描述

INSERT INTO命令将表中的数据插入到已关联的指定DWS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**SELECT基本语句**。

参数说明

表 1-54 参数描述

参数	描述
DLI_TABLE	已创建跨源连接的表名称。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

- 表必须已经存在。
- 表在创建时不需要指定Schema信息，Schema信息将使用DWS表的信息。如果select子句中选择的字段数量和类型与DWS表的Schema信息不匹配时，系统将报错。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

1.18.3 查询 DWS 表

SELECT命令用于查询DWS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表dli_to_dws中的数据。

```
SELECT * FROM dli_to_dws limit 100;
```

1.19 跨源连接 RDS 表

1.19.1 创建表关联 RDS

功能描述

使用CREATE TABLE命令创建表并关联RDS上已有的表。该功能支持访问RDS的MySQL集群和PostGre集群。

说明

Spark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提供的跨源认证方式。

前提条件

创建表关联RDS之前需要创建跨源连接。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME
  USING JDBC OPTIONS (
  'url'='xx',
  'driver'='DRIVER_NAME',
  'dbtable'='db_name_in_RDS.table_name_in_RDS',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

关键字

表 1-55 CREATE TABLE 关键字说明

参数	描述
url	RDS的连接地址，需要先创建跨源连接。 创建增强型跨源连接后，使用RDS提供的"内网域名"或者内网地址和数据库端口访问，MySQL格式为"协议头://内网IP:内网端口"，PostGre格式为"协议头://内网IP:内网端口/数据库名"。 例如："jdbc:mysql://192.168.0.193:3306"或者"jdbc:postgresql://192.168.0.193:3306/postgres"。
driver	jdbc驱动类名，访问MySQL集群请填写："com.mysql.jdbc.Driver"，访问PostGre集群请填写："org.postgresql.Driver"。
dbtable	• 访问MySQL集群填写"数据库名.表名" **注意** 连接的RDS数据库名不能包含中划线-或^特殊字符，否则会创建表失败。 • 访问PostGre集群填写"模式名.表名" **说明** 模式名即为数据库模式（schema）的名称。数据库中schema是数据库对象集合，包含了表，视图等多种对象。
user	（已废弃）RDS用户名。
password	（已废弃）RDS用户名密码。
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>。
encryption	使用跨源密码认证时配置为“true”。

参数	描述
partitionColumn	读取数据时，用于设置并发使用的数值型字段。 **说明** • “partitionColumn”、“lowerBound”、“upperBound”、“numPartitions”四个参数必须同时设置，不支持仅设置其中某一个或某几个。 • 为了提升并发读取的性能，建议使用自增列。
lowerBound	partitionColumn设置的字段数据最小值，该值包含在返回结果中。
upperBound	partitionColumn设置的字段数据最大值，该值不包含在返回结果中。

numPartitions	读取数据时并发数。
	说明
	实际读取数据时，会根据“lowerBound”与“upperBound”，平均分配给每个task，获取其中一部分的数据。例如：
	`'partitionColumn'='id',`
	`'lowerBound'='0',`
	`'upperBound'='100',`
	`'numPartitions'='2'`
	表示在DLI中会起2个并发task，一个task执行id>=0 and id < 50，另一个task执行id >=50 and id < 100。

fetchsize	读取数据时，每一批次获取数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
batchsize	写入数据时，每一批次写入数据的记录数，默认值1000。设置越大性能越好，但占用内存越多，该值设置过大会有内存溢出的风险。
truncate	执行overwrite时是否不删除原表，直接执行清空表操作，取值范围： • true • false 默认为“false”，即在执行overwrite操作时，先将原表删除再重新建表。
isolationLevel	事务隔离级别，取值范围： • NONE • READ_UNCOMMITTED • READ_COMMITTED • REPEATABLE_READ • SERIALIZABLE 默认值为“READ_UNCOMMITTED”。

注意事项

创建RDS关联表时，不需要指定关联表的Schema。DLI会自动获取RDS中对应参数"dbtable"中的表的Schema。

示例

访问MySQL

```
CREATE TABLE IF NOT EXISTS dli_to_rds
  USING JDBC OPTIONS (
  'url'='jdbc:mysql://to-rds-117405104-3eAHxnlz.datasource.com:3306',
  'driver'='com.mysql.jdbc.Driver',
  'dbtable'='rds_test.test1',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

访问PostGre

```
CREATE TABLE IF NOT EXISTS dli_to_rds
  USING JDBC OPTIONS (
  'url'='jdbc:postgresql://to-rds-1174405119-oLRHAGE7.datasource.com:3306/postgreDB',
  'driver'='org.postgresql.Driver',
  'dbtable'='pg_schema.test1',
  'passwdauth' = 'xxx',
  'encryption' = 'true');
```

1.19.2 插入数据至 RDS 表

功能描述

INSERT INTO命令将表中的数据插入到已关联的指定RDS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**SELECT基本语句**。

参数说明

表 1-56 参数描述

参数	描述
DLI_TABLE	已创建跨源连接的表名称。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。

参数	描述
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

- 表必须已经存在。
- 表在创建时不需要指定Schema信息，Schema信息将使用RDS表的信息。如果select子句中选择的字段数量和类型与RDS表的Schema信息不匹配时，系统将报错。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

1.19.3 查询 RDS 表

SELECT命令用于查询RDS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表test_ct中的数据。

```
SELECT * FROM dli_to_rds limit 100;
```

1.20 跨源连接 CSS 表

1.20.1 创建表关联 CSS

功能描述

使用CREATE TABLE命令创建表并关联CSS上已有的表。

说明

Spark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提供的跨源认证方式。

前提条件

创建表关联CSS之前需要创建跨源连接。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
  FIELDNAME1 FIELDTYPE1,
  FIELDNAME2 FIELDTYPE2)
  USING CSS OPTIONS (
  'es.nodes'='xx',
  'resource'='type_path_in_CSS',
  'pushdown'='true',
  'strict'='false',
  'batch.size.entries'= '1000',
  'batch.size.bytes'= '1mb',
  'es.nodes.wan.only' = 'true',
  'es.mapping.id' = 'FIELDNAME');
```

关键字

表 1-57 CREATE TABLE 关键字说明

参数	描述
es.nodes	CSS的连接地址，需要先创建跨源连接。 创建增强型跨源连接后，使用CSS提供的"内网访问地址"，格式为"IP1:PORT1,IP2:PORT2"。
resource	指定在CSS关联的资源名，用"/index/type"指定资源位置（可简单理解index为database，type为table，但绝不等同）。 **说明** • ES 6.X版本中，单个Index只支持唯一type，type名可以自定义。 • ES 7.X版本中，单个Index将使用“_doc”作为type名，不再支持自定义。若访问ES 7.X版本时，该参数只需要填写index即可。
pushdown	CSS的下压功能是否开启，默认为“true”。包含大量IO传输的表在有where过滤条件的情况下能够开启pushdown降低IO。

参数	描述
strict	CSS的下压是否是严格的，默认为“false”。精确匹配的场景下比pushdown降低更多IO。
batch.size.entries	单次batch插入entry的条数上限，默认为1000。如果单条数据非常大，在bulk存储设置的数据条数前提前到达了单次batch的总数据量上限，则停止存储数据，以batch.size.bytes为准，提交该批次的数据。
batch.size.bytes	单次batch的总数据量上限，默认为1mb。如果单条数据非常小，在bulk存储到总数据量前提前到达了单次batch的条数上限，则停止存储数据，以batch.size.entries为准，提交该批次的数据。
es.nodes.wan.only	是否仅通过域名访问es节点，默认为false。使用css服务提供的原始内网IP地址作为es.nodes时，不需要填写该参数或者配置为false。
es.mapping.id	指定一个字段，其值作为es中Document的id。 **说明** • 相同/index/type下的Document id是唯一的。如果作为Document id的字段存在重复值，则在执行插入es时，重复id的Document将会被覆盖。 • 该特性可以用作容错解决方案。当插入数据执行一半时，DLI作业失败，会有部分数据已经插入到es中，这部分为冗余数据。如果设置了Document id，则在重新执行DLI作业时，会覆盖上一次的冗余数据。
es.net.ssl	连接安全CSS集群，默认值为false
es.certificate.name	连接安全CSS集群，使用的跨源认证信息名称。跨源认证信息创建方式请参考《 数据湖探索用户指南 》>。

说明

batch.size.entries和batch.size.bytes分别对数据条数和数据量大小进行限制。

示例

```
CREATE TABLE IF NOT EXISTS dli_to_css (doc_id String, name string, age int)
  USING CSS OPTIONS (
  es.nodes 'to-css-1174404703-LzwpJEyx.datasource.com:9200',
  resource '/dli_index/dli_type',
  pushdown 'false',
  strict 'true',
  es.nodes.wan.only 'true',
  es.mapping.id 'doc_id');
```

1.20.2 插入数据至 CSS 表

功能描述

INSERT INTO命令将表中的数据插入到已关联的指定CSS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
```



```
[FROM DLI_TEST]
[WHERE where_condition]
[LIMIT num]
[GROUP BY field]
[ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**SELECT基本语句**。

参数说明

表 1-58 参数描述

参数	描述
DLI_TABLE	已创建跨源连接的表名称。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

- 表必须已经存在。
- 表在创建时需要指定Schema信息，如果select子句或者values中字段数量与CSS表的Schema字段数量不匹配时，系统将报错。
- 类型不一致时不一定报错，例如插入int类型数据，但CSS中Schema保存的是文本类型，int类型会被转换成文本类型。
- 不建议对同一张表并发插入数据，因为有一定概率发生并发冲突，导致插入失败。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

1.20.3 查询 CSS 表

SELECT命令用于查询CSS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

所查询的表必须是已经存在的表，否则会出错。

示例

查询表dli_to_css中的数据。

```
SELECT * FROM dli_to_css limit 100;
```

1.21 跨源连接 DCS 表

1.21.1 创建表关联 DCS

功能描述

使用CREATE TABLE命令创建表并关联DCS上已有的Key。

说明

Spark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提供的跨源认证方式。

前提条件

创建表关联DCS之前需要创建跨源连接，绑定队列。

语法格式

- 指定Key

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
    FIELDNAME1 FIELDTYPE1,
    FIELDNAME2 FIELDTYPE2)
  USING REDIS OPTIONS (
  'host'='xx',
  'port'='xx',
  'passwdauth' = 'xxx',
  'encryption' = 'true',
  'table'='namespace_in_redis:key_in_redis',
  'key.column'= 'FIELDNAME1'
);
```

- 通配key

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
   FIELDNAME1 FIELDTYPE1,
   FIELDNAME2 FIELDTYPE2)
 USING REDIS OPTIONS (
 'host'='xx',
 'port'='xx',
 'passwdauth' = 'xxx',
 'encryption' = 'true',
 'keys.pattern'='key*:*',
 'key.column'= 'FIELDNAME1'
);
```

关键字

表 1-59 CREATE TABLE 关键字说明

参数	描述
host	DCS的连接IP，需要先创建跨源连接。 创建增强型跨源连接后，使用DCS提供的"连接地址"。"连接地址"有多个时，选择其中一个即可。 **说明** 访问DCS目前只支持增强型跨源。
port	DCS的连接端口，例如6379。
password	（已废弃）创建DCS集群时填写的密码。访问非安全Redis集群时不需要填写。
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>。
encryption	使用跨源密码认证时配置为“true”。
table	对应Redis中的Key或Hash Key。 ● 插入redis数据时必填。 ● 查询redis数据时与“keys.pattern”参数二选一。
keys.pattern	使用正则表达式匹配多个Key或Hash Key。该参数仅用于查询时使用。查询redis数据时与“table”参数二选一。
key.column	非必填。用于指定schema中的某个字段作为Redis中key的标识。在插入数据时与参数“table”配合使用。
partitions.number	读取数据时，并发task数。
scan.count	每批次读取的数据记录数，默认为100。如果在读取过程中，redis集群中的CPU使用率还有提升空间，可以调大该参数。
iterator.grouping.size	每批次插入的数据记录数，默认为100。如果在插入过程中，redis集群中的CPU使用率还有提升空间，可以调大该参数。
timeout	连接redis的超时时间，单位ms，默认值2000（2秒超时）。

说明

访问DCS时，不支持复杂类型数据（Array、Struct、Map等）。

可以考虑以下几种方式进行复杂类型数据处理：

- 字段扁平化处理，将下一级的字段展开放在同一层Schema字段中。
- 使用二进制方式进行写入与读取，并通过自定义函数进行编解码。

示例

- 指定table

```
create table test_redis(name string, age int) using redis options(
  'host' = '192.168.4.199',
  'port' = '6379',
  'passwdauth' = 'xxx',
  'encryption' = 'true',
  'table' = 'person'
);
```

- 通配table名

```
create table test_redis_keys_patten(id string, name string, age int) using redis options(
  'host' = '192.168.4.199',
  'port' = '6379',
  'passwdauth' = 'xxx',
  'encryption' = 'true',
  'keys.pattern' = 'p*:*',
  'key.column' = 'id'
);
```

1.21.2 插入数据至 DCS 表

功能描述

INSERT INTO命令将表中的数据插入到已关联的DCS Key中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

关键字

SELECT对应关键字说明请参考**SELECT基本语句**。

参数说明

表 1-60 参数描述

参数	描述
DLI_TABLE	已创建跨源连接的表名称。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

- 表必须已经存在。
- 表在创建时需要指定Schema信息。
- 如果在建表时指定“key.column”，则在Redis中会以指定字段的值作为Redis Key名称的一部分。例如：

```
create table test_redis(name string, age int) using redis options(
  'host' = '192.168.4.199',
  'port' = '6379',
  'passwdauth' = '******',
  'table' = 'test_with_key_column',
  'key.column' = 'name'
);
insert into test_redis values("James", 35), ("Michael", 22);
```

在redis中将会有2个名为test_with_key_column:James和test_with_key_column:Michael的表：

```
192.168.7.238:6379> keys test_with_key_column:*
1) "test_with_key_column:Michael"
2) "test_with_key_column:James"
192.168.7.238:6379>
```

```
192.168.7.238:6379> hgetall "test_with_key_column:Michael"
1) "age"
2) "22"
192.168.7.238:6379> hgetall "test_with_key_column:James"
1) "age"
2) "35"
192.168.7.238:6379>
```

- 如果在建表时没有指定“key.column”，则在Redis中的key name将会使用uuid。例如：

```
create table test_redis(name string, age int) using redis options(
  'host' = '192.168.7.238',
  'port' = '6379',
  'passwdauth' = '******',
  'table' = 'test_without_key_column'
);
insert into test_redis values("James", 35), ("Michael", 22);
```

在redis中将会有2个以“test_without_key_column:uuid”命名的表：

```
192.168.7.238:6379> keys test_without_key_column:*
1) "test_without_key_column:b0ce581fa0d548e5b2273f4db1df6dcd"
2) "test_without_key_column:1e80aa7175d747ee9a82cce241767b01"
192.168.7.238:6379>
```

```
192.168.7.238:6379> hgetall "test_without_key_column:b0ce581fa0d548e5b2273f4db1df6dcd"
1) "age"
2) "35"
3) "name"
4) "James"
192.168.7.238:6379> hgetall "test_without_key_column:1e80aa7175d747ee9a82cce241767b01"
1) "age"
2) "22"
3) "name"
4) "Michael"
192.168.7.238:6379>
```

示例

```
INSERT INTO test_redis
  VALUES("James", 35), ("Michael", 22);
```

1.21.3 查询 DCS 表

SELECT命令用于查询DCS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

示例

查询表test_redis中的数据。

```
SELECT * FROM test_redis limit 100;
```

1.22 跨源连接 DDS 表

1.22.1 创建表关联 DDS

功能描述

使用CREATE TABLE命令创建表并关联DDS上已有的collection。

说明

Spark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提供的跨源认证方式。

前提条件

创建表关联DDS之前需要创建跨源连接，绑定队列。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME(
   FIELDNAME1 FIELDTYPE1,
   FIELDNAME2 FIELDTYPE2)
  USING MONGO OPTIONS (
  'url'='IP:PORT[,IP:PORT]/[DATABASE][.COLLECTION][AUTH_PROPERTIES]',
  'database'='xx',
  'collection'='xx',
  'passwdauth' = 'xxx',
  'encryption' = 'true'
);
```

说明

文档数据库服务（Document Database Service，简称DDS）完全兼容MongoDB协议，因此语法中使用“using mongo options”。

关键字

表 1-61 CREATE TABLE 关键字说明

参数	描述
url	DDS的连接信息，需要先创建跨源连接 创建增强型跨源连接后，使用DDS提供的"随机连接地址"，格式为： "IP:PORT[,IP:PORT]/[DATABASE][.COLLECTION][AUTH_PROPERTIES]" 例如："192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin"
database	DDS的数据库名，如果在"url"中同时指定了数据库名，则"url"中的数据库名不生效。
collection	DDS中的collection名，如果在"url"中同时指定了collection，则"url"中的collection不生效。
user	（已废弃）访问DDS集群用户名。
password	（已废弃）访问DDS集群密码
passwdauth	跨源密码认证名称。跨源认证信息创建方式请参考《数据湖探索用户指南》>。
encryption	使用跨源密码认证时配置为“true”。

说明

如果在DDS中已存在collection，则建表可以不指定schema信息，DLI会根据collection中的数据自动生成schema信息。

示例

```
create table 1_datasource_mongo.test_momgo(id string, name string, age int) using mongo options(
  'url' = '192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin',
  'database' = 'test',
  'collection' = 'test',
```

```
'passwdauth' = 'xxx',
'encryption' = 'true');
```

1.22.2 插入数据至 DDS 表

功能描述

INSERT INTO命令将表中的数据插入到已关联的指定DDS表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

- 覆盖插入数据

```
INSERT OVERWRITE TABLE DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

关键字

SELECT对应关键字说明请参考**SELECT基本语句**。

参数说明

表 1-62 参数描述

参数	描述
DLI_TABLE	已创建跨源连接的表名称。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

表必须已经存在。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
  SELECT ATTR_EXPR
  FROM user
  WHERE user_name='cyz'
  LIMIT 3
  GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
  VALUES (1);
```

1.22.3 查询 DDS 表

SELECT命令用于查询DDS表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

如果在建表时没有指定schema信息，则查询出来的结果将会包含"_id"字段用于存放doc中的"_id"。

示例

查询表test_table1中的数据。

```
SELECT * FROM test_table1 limit 100;
```

1.23 跨源连接 Oracle 表

1.23.1 创建表关联 Oracle

功能描述

使用CREATE TABLE命令创建表并关联Oracle上已有的表。

前提条件

- 创建DLI表关联Oracle之前需要创建增强型跨源连接。

语法格式

```
CREATE TABLE [IF NOT EXISTS] TABLE_NAME
  USING ORACLE OPTIONS (
  'url'='xx',
  'driver'='DRIVER_NAME',
  'dbtable'='db_in_oracle.table_in_oracle',
  'user' = 'xxx',
  'password' = 'xxx',
  'resource' = 'obs://rest-authinfo/tools/oracle/driver/ojdbc6.jar'
);
```

关键字

表 1-63 CREATE TABLE 关键字说明

参数	描述
url	Oracle的连接地址。 Oracle url支持以下格式： ● 格式一：jdbc:oracle:thin:@host:port:SID，其中SID是oracle数据库的唯一标识符。 ● 格式二：jdbc:oracle:thin:@//host:port/service_name；这种方式是Oracle推荐的，对于集群来说，每个节点的SID可能不一致，但ServiceName是一致的，包含所有节点。
driver	Oracle驱动类名: oracle.jdbc.driver.OracleDriver
dbtable	指定在Oracle关联的表名，或者"用户名.表名"，例如：public.table_name。
user	Oracle用户名。
password	Oracle用户名密码。
resource	Oracle驱动包的OBS路径。 例如：obs://rest-authinfo/tools/oracle/driver/ojdbc6.jar resource中定义的driver jar包如果被更新，需要重启队列，才会生效。

示例

创建Oracle跨源表

```
CREATE TABLE IF NOT EXISTS oracleTest
  USING ORACLE OPTIONS (
  'url'='jdbc:oracle:thin:@//192.168.168.40:1521/helowin',
  'driver'='oracle.jdbc.driver.OracleDriver',
  'dbtable'='test.Student',
  'user' = 'test',
  'password' = 'test',
  'resource' = 'obs://rest-authinfo/tools/oracle/driver/ojdbc6.jar'
);
```

1.23.2 插入数据至 Oracle 表

功能描述

INSERT INTO命令将数据插入到已关联的指定Oracle表中。

语法格式

- 将SELECT查询结果插入到表中：

```
INSERT INTO DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

- 将某条数据插入到表中：

```
INSERT INTO DLI_TABLE
  VALUES values_row [, values_row ...];
```

- 覆盖插入数据

```
INSERT OVERWRITE TABLE DLI_TABLE
  SELECT field1,field2...
  [FROM DLI_TEST]
  [WHERE where_condition]
  [LIMIT num]
  [GROUP BY field]
  [ORDER BY field] ...;
```

关键字

SELECT对应关键字说明请参考**SELECT基本语句**。

参数说明

表 1-64 参数描述

参数	描述
DLI_TABLE	已创建跨源连接的表名称。
DLI_TEST	为包含待查询数据的表。
field1,field2...，field	表“DLI_TEST”中的列值，需要匹配表“DLI_TABLE”的列值和类型。
where_condition	查询过滤条件。
num	对查询结果进行限制，num参数仅支持INT类型。
values_row	想要插入到表中的值，列与列之间用逗号分隔。

注意事项

表必须已经存在。

示例

- 查询表“user”中的数据插入表“test”中。

```
INSERT INTO test
 SELECT ATTR_EXPR
 FROM user
 WHERE user_name='cyz'
 LIMIT 3
 GROUP BY user_age
```

- 插入数据“1”到表“test”中

```
INSERT INTO test
 VALUES (1);
```

1.23.3 查询 Oracle 表

功能描述

SELECT命令用于查询Oracle表中的数据。

语法格式

```
SELECT * FROM table_name LIMIT number;
```

关键字

LIMIT：对查询结果进行限制，number参数仅支持INT类型。

注意事项

如果在建表时没有指定schema信息，则查询出来的结果将会包含"_id"字段用于存放doc中的"_id"。

示例

查询表test_oracle中的数据。

```
SELECT * FROM test_oracle limit 100;
```

1.24 视图

1.24.1 创建视图

功能描述

创建视图。

语法格式

```
CREATE [OR REPLACE] VIEW view_name AS select_statement;
```

关键字

- CREATE VIEW：基于给定的select语句创建视图，不会将select语句的结果写入磁盘。

- OR REPLACE：指定该关键字后，若视图已经存在将不报错，并根据select语句更新视图的定义。

注意事项

- 所要创建的视图必须是当前数据库下不存在的，否则会报错。当视图存在时，可通过增加OR REPLACE关键字来避免报错。
- 视图中包含的表或视图信息不可被更改，如有更改可能会造成查询失败。
- 如果创建表和创建视图使用的计算引擎不一致，可能会因为varchar类型不兼容，导致视图查询失败。

  例如：使用Spark 3.x版本创建的表，建议您使用Spark 2.x创建相应的视图。

示例

先通过对student表中的id和name数据进行查询，并以该查询结果创建视图student_view。

```
CREATE VIEW student_view AS SELECT id, name FROM student;
```

1.24.2 删除视图

功能描述

删除视图。

语法格式

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

关键字

DROP：删除指定视图的元数据。虽然视图和表有很多共同之处，但是DROP TABLE不能用来删除VIEW。

注意事项

所要删除的视图必须是已经存在的，否则会出错，可以通过IF EXISTS来避免该错误。

示例

删除名为student_view的视图。

```
DROP VIEW student_view;
```

1.25 查看计划

功能描述

执行该语句将返回该SQL语句的逻辑计划与物理执行计划。

语法格式

```
EXPLAIN [EXTENDED | CODEGEN] statement;
```

关键字

EXTENDED：指定该关键字后，会同时输出逻辑计划与物理执行计划。

CODEGEN：指定该关键字后，若有codegen产生的代码也将输出。

注意事项

无。

示例

返回“SELECT * FROM test”SQL语句的逻辑计划与物理执行计划。

```
EXPLAIN EXTENDED select * from test;
```

1.26 数据权限管理

1.26.1 数据权限列表

DLI中SQL语句与数据库、表、角色相关的权限矩阵如表1-65所示。

表 1-65 权限矩阵

分类	SQL语句	权限	说明
Database	DROP DATABASE db1	database.db1的DROP_DATABASE权限	-
	CREATE TABLE tb1(...)	database.db1的CREATE_TABLE权限	-
	CREATE VIEW v1	database.db1的CREATE_VIEW权限	-
	EXPLAIN query	database.db1的EXPLAIN权限	query需要其相应的权限。
Table	SHOW CREATE TABLE tb1	database.db1.tables.tb1的SHOW_CREATE_TABLE权限	-
	DESCRIBE [EXTENDED| FORMATTED] tb1	databases.db1.tables.tb1的DESCRIBE_TABLE权限	-
	DROP TABLE [IF EXISTS] tb1	database.db1.tables.tb1的DROP_TABLE权限	-
	SELECT * FROM tb1	database.db1.tables.tb1的SELECT权限	-
	SELECT count(*) FROM tb1	database.db1.tables.tb1的SELECT权限	-

分类	SQL语句	权限	说明
	SELECT * FROM view1	database.db1.tables.view1的SELECT权限	-
	SELECT count(*) FROM view1	database.db1.tables.view1的SELECT权限	-
	LOAD DLI TABLE	database.db1.tables.tb1的INSERT_INTO_TABLE权限	-
	INSERT INTO TABLE	database.db1.tables.tb1的INSERT_INTO_TABLE权限	-
	INSERT OVERWRITE TABLE	database.db1.tables.tb1的INSERT_OVERWRITE_TABLE权限	-
	ALTER TABLE ADD COLUMNS	database.db1.tables.tb1的ALTER_TABLE_ADD_COLUMNS权限	-
	ALTER TABLE RENAME	database.db1.tables.tb1的ALTER_TABLE_RENAME权限	-
ROLE&PRIVILEGE	CREATE ROLE	db的CREATE_ROLE权限	-
	DROP ROLE	db的DROP_ROLE权限	-
	SHOW ROLES	db的SHOW_ROLES权限	-
	GRANT ROLES	db的GRANT_ROLE权限	-
	REVOKE ROLES	db的REVOKE_ROLE权限	-
	GRANT PRIVILEGE	db或table的GRANT_PRIVILEGE权限	-
	REVOKE PRIVILEGE	db或table的REVOKE_PRIVILEGE权限	-
	SHOW GRANT	db或table的SHOW_GRANT权限	-

Privilege在进行数据库和表赋权或回收权限时，DLI支持的权限类型如下所示。

- DATABASE上可赋权/回收的权限：
  - DROP_DATABASE（删除数据库）
  - CREATE_TABLE（创建表）
  - CREATE_VIEW（创建视图）
  - EXPLAIN（将SQL语句解释为执行计划）
  - CREATE_ROLE（创建角色）
  - DROP_ROLE（删除角色）

  - SHOW_ROLES（显示角色）
  - GRANT_ROLE（绑定角色）
  - REVOKE_ROLE（解除角色绑定）
  - DESCRIBE_TABLE（描述表）
  - DROP_TABLE（删除表）
  - SELECT（查询表）
  - INSERT_INTO_TABLE（插入）
  - INSERT_OVERWRITE_TABLE（重写）
  - GRANT_PRIVILEGE（数据库的赋权）
  - REVOKE_PRIVILEGE（数据库权限的回收）
  - SHOW_PRIVILEGES（查看其他用户具备的数据库权限）
  - ALTER_TABLE_ADD_PARTITION（在分区表中添加分区）
  - ALTER_TABLE_DROP_PARTITION（删除分区表的分区）
  - ALTER_TABLE_RENAME_PARTITION（重命名表分区）
  - ALTER_TABLE_RECOVER_PARTITION（恢复表分区）
  - ALTER_TABLE_SET_LOCATION（设置分区的路径）
  - SHOW_PARTITIONS（显示所有分区）
  - SHOW_CREATE_TABLE（查看建表语句）
- TABLE上可以赋权/回收的权限：
  - DESCRIBE_TABLE（描述表）
  - DROP_TABLE（删除表）
  - SELECT（查询表）
  - INSERT_INTO_TABLE（插入）
  - INSERT_OVERWRITE_TABLE（重写）
  - GRANT_PRIVILEGE（表的赋权）
  - REVOKE_PRIVILEGE（表权限的回收）
  - SHOW_PRIVILEGES（查看其他用户具备的表权限）
  - ALTER_TABLE_ADD_COLUMNS（增加列）
  - ALTER_TABLE_RENAME（重命名表）
  - ALTER_TABLE_ADD_PARTITION（在分区表中添加分区）
  - ALTER_TABLE_DROP_PARTITION（删除分区表的分区）
  - ALTER_TABLE_RENAME_PARTITION（重命名表分区）
  - ALTER_TABLE_RECOVER_PARTITION（恢复表分区）
  - ALTER_TABLE_SET_LOCATION（设置分区的路径）
  - SHOW_PARTITIONS（显示所有分区）
  - SHOW_CREATE_TABLE（查看建表语句）

1.26.2 创建角色

功能描述

- 在当前database或指定database中创建一个新的角色。

- 只有在database上具有CREATE_ROLE权限的用户才能创建角色。例如：管理员用户、database的owner用户和被赋予了CREATE_ROLE权限的其他用户。
- 每个角色必须属于且只能属于一个database。

语法格式

```
CREATE ROLE [db_name].role_name;
```

关键字

无。

注意事项

- 要创建的role_name必须在当前database或指定database中不存在，否则会报错。
- 当未指定“db_name”时，表示在当前database中创建角色。

示例

```
CREATE ROLE role1;
```

1.26.3 删除角色

功能描述

在当前database或指定database中删除角色。

语法格式

```
DROP ROLE [db_name].role_name;
```

关键字

无。

注意事项

- 要删除的role_name必须在当前database或指定database中存在，否则会报错。
- 当未指定“db_name”时，表示在当前database中删除角色。

示例

```
DROP ROLE role1;
```

1.26.4 绑定角色

功能描述

绑定用户和角色。

语法格式

```
GRANT ([db_name].role_name,...) TO (user_name,...);
```

关键字

无。

注意事项

role_name和username必须存在，否则会报错。

示例

```
GRANT role1 TO user_name1;
```

1.26.5 解绑角色

功能描述

取消用户和角色的绑定。

语法格式

```
REVOKE ([db_name].role_name,...) FROM (user_name,...);
```

关键字

无。

注意事项

role_name和user_name必须存在，且user_name绑定了该role_name。

示例

取消用户user_name1和role1的绑定。

```
REVOKE role1 FROM user_name1;
```

1.26.6 显示角色

功能描述

显示所有的角色或者显示当前database下绑定到“user_name”的角色。

语法格式

```
SHOW [ALL] ROLES [user_name];
```

关键字

ALL：显示所有的角色。

注意事项

ALL关键字与user_name不可同时存在。

示例

- 显示绑定到该用户的所有角色。
  ```
  SHOW ROLES;
  ```
- 显示project下的所有角色。
  ```
  SHOW ALL ROLES;
  ```

  说明

  只有管理员才有权限执行show all roles语句。
- 显示绑定到用户名为user_name1的所有角色。
  ```
  SHOW ROLES user_name1;
  ```

1.26.7 分配权限

功能描述

授予用户或角色权限。

语法格式

```
GRANT (privilege,...) ON (resource,..) TO ((ROLE [db_name].role_name) | (USER user_name)),...);
```

关键字

ROLE：限定后面的role_name是一个角色。

USER：限定后面的user_name是一个用户。

注意事项

- privilege必须是可授权限中的一种。且如果赋权对象在resource或上一级resource上已经有对应权限时，则会赋权失败。Privilege支持的权限类型可参见**数据权限列表**。
- resource可以是queue、database、table、view、column，格式分别为：
  - queue的格式为：queues.queue_name

    queue支持的Privilege权限类型可以参考下表：

操作	说明
DROP_QUEUE	删除队列
SUBMIT_JOB	提交作业
CANCEL_JOB	终止作业
RESTART	重启队列
SCALE_QUEUE	扩缩容队列
GRANT_PRIVILEGE	队列的赋权
REVOKE_PRIVILEGE	队列权限的回收
SHOW_PRIVILEGES	查看其他用户具备的队列权限

- database的格式为：databases.db_name
  database支持的Privilege权限类型可参见**数据权限列表**。
- table的格式为：databases.db_name.tables.table_name
  table支持的Privilege权限类型可参见**数据权限列表**。
- view的格式为：databases.db_name.tables.view_name
  view支持的Privilege权限类型和table一样，具体可以参考**数据权限列表**中table的权限列表描述。
- column的格式为：
  databases.db_name.tables.table_name.columns.column_name
  column支持的Privilege权限类型仅为：SELECT

示例

给用户user_name1授予数据库db1的删除数据库权限。

```
GRANT DROP_DATABASE ON databases.db1 TO USER user_name1;
```

给用户user_name1授予数据库db1的表tb1的SELECT权限。

```
GRANT SELECT ON databases.db1.tables.tb1 TO USER user_name1;
```

给角色role_name授予数据库db1的表tb1的SELECT权限。

```
GRANT SELECT ON databases.db1.tables.tb1 TO ROLE role_name;
```

1.26.8 回收权限

功能描述

回收已经授予用户或角色的权限。

语法格式

```
REVOKE (privilege,...) ON (resource,..) FROM ((ROLE [db_name].role_name) | (USER user_name)),...);
```

关键字

ROLE：限定后面的rol e_name是一个角色。

USER：限定后面的user_name是一个用户。

注意事项

- privilege必须为赋权对象在resource中的已授权限，否则会回收失败。Privilege支持的权限类型可参见**数据权限列表**。
- resource可以是queue、database、table、view、column，格式分别为：
  - queue的格式为：queues.queue_name
  - database的格式为：databases.db_name
  - table的格式为：databases.db_name.tables.table_name
  - view的格式为：databases.db_name.tables.view_name
  - column的格式为：
    databases.db_name.tables.table_name.columns.column_name

示例

回收用户user_name1对于数据库db1的删除数据库权限。

```
REVOKE DROP_DATABASE ON databases.db1 FROM USER user_name1;
```

回收用户user_name1对于数据库db1的表tb1的SELECT权限。

```
REVOKE SELECT ON databases.db1.tables.tb1 FROM USER user_name1;
```

回收角色role_name对于数据库db1的表tb1的SELECT权限。

```
REVOKE SELECT ON databases.db1.tables.tb1 FROM ROLE role_name;
```

1.26.9 显示已授权限

功能描述

显示某个用户在resource上已经授予的权限。

语法格式

```
SHOW GRANT USER user_name ON resource;
```

关键字

USER：限定后面的user_name是一个用户。

注意事项

resource可以是queue、database、table、column、view，格式分别为：

- queue的格式为：queues.queue_name
- database的格式为：databases.db_name
- table的格式为：databases.db_name.tables.table_name
- column的格式为：databases.db_name.tables.table_name.columns.column_name
- view的格式为：databases.db_name.tables.view_name

示例

显示用户user_name1在数据库db1上的权限。

```
SHOW GRANT USER user_name1 ON databases.db1;
```

1.26.10 显示所有角色和用户的绑定关系

功能描述

在当前database显示角色与某用户的绑定关系。

语法格式

```
SHOW PRINCIPALS ROLE;
```

关键字

无。

注意事项

变量ROLE必须存在。

示例

```
SHOW PRINCIPALS role1;
```

1.27 数据类型

1.27.1 概述

数据类型是数据的一个基本属性，用于区分不同类型的数据。不同的数据类型所占的存储空间不同，能够进行的操作也不相同。数据库中的数据存储在表中。表中的每一列都定义了数据类型，用户存储数据时，须遵从这些数据类型的属性，否则可能会出错。

DLI当前只支持原生数据类型。

1.27.2 原生数据类型

DLI支持原生数据类型，请参见**表1-66**。

表 1-66 原生数据类型

数据类型	描述	存储空间	范围	OBS表支持情况
INT	有符号整数	4字节	-2147483648 ~ 2147483647	是
STRING	字符串	-	-	是
FLOAT	单精度浮点型	4字节	-	是
DOUBLE	双精度浮点型	8字节	-	是
DECIMAL(precision,scale)	10进制精确数字类型。固定有效位数和小数位数的数据类型，例如：3.5 • precision：表示最多可以表示多少位的数字。 • scale：表示小数部分的位数。	-	1<=precision<=38 0<=scale<=38 若不指定precision和scale，则默认为decimal（38,38）。	是
BOOLEAN	布尔类型	1字节	TRUE/FALSE	是

数据类型	描述	存储空间	范围	OBS表支持情况
SMALLINT/ SHORT	有符号整数	2字节	-32768~32767	是
TINYINT	有符号整数	1字节	-128~127	是
BIGINT/ LONG	有符号整数	8字节	-9223372036854775808～92233720368 54775807	是
TIMESTAMP	时间戳，表示日期和时间，格式为原始数据。例如：1621434131222	-	-	是
CHAR	固定长度字符串	-	-	是
VARCHAR	可变长度字符串	-	-	是
DATE	日期类型，描述了特定的年月日，以yyyy-mm-dd格式表示，例如：2014-05-29	-	DATE类型不包含时间，所表示日期的范围为0000-01-01~9999-12-31。	是

说明

- VARCHAR和CHAR在DLI实际存储是STRING型，因此超出长度的字符串不会被截断。
- FLOAT类型在DLI实际存储是DOUBLE型。

INT

有符号整数，存储空间为4字节，-2147483648～2147483647，在NULL情况下，默认值为0。

STRING

字符串类型。

FLOAT

单精度浮点型，存储空间为4字节，在NULL情况下，采用计算值默认值为0。

由于浮点类型的数据在计算机中的存储方式的限制，在比较两个浮点类型的数据是否相等时，因存在精度问题，不能直接采用“a==b”的方式进行比较，建议使用“(a-b)的绝对值<=EPSILON”这种方式进行比较，EPSILON为允许的误差范围，一般为1.19209290E-07F。若两个浮点数的差值的绝对值在这个范围内就认为相等。

DOUBLE

双精度浮点型，存储空间为8字节，在NULL情况下，采用计算值默认值为0。

由于浮点类型的数据在计算机中的存储方式的限制，在比较两个浮点类型的数据是否相等时，因存在精度问题，不能直接采用“a==b”的方式进行比较，建议使用“(a-b)的绝对值<=EPSILON”这种方式进行比较，EPSILON为允许的误差范围，一般为2.2204460492503131E-16。若两个浮点数的差值的绝对值在这个范围内就认为相等。

DECIMAL

Decimal(p,s)表示数值中共有p位数，其中整数p-s位，小数s位。p表示可储存的最大十进制数的位数总数，小数点左右两侧都包括在内。有效位数p必须是1至最大有效位数38之间的值。s表示小数点右侧所能储存的最大十进制数的位数。小数位数必须是从0到p的值。只有在指定了有效位数时，才能指定小数位数。因此，0 ≤ s ≤ p。例如：decimal(10,6)，表示数值中共有10位数，其中整数占4位，小数占6位。

BOOLEAN

布尔类型，包括TRUE与FALSE。

SMALLINT/SHORT

有符号整数，存储空间为2字节，范围为-32768～32767。当为NULL情况下，采用计算值默认为0。

TINYINT

有符号整数，存储空间为1字节，范围为-128～127。当为NULL情况下，采用计算值默认为0。

BIGINT/LONG

有符号整数，存储空间为8字节，范围为-9223372036854775808～9223372036854775807，不支持科学计数法。当为NULL情况下。采用计算值默认为0。

TIMESTAMP

支持传统的UNIX TIMESTAMP，提供达到微秒级别精度的选择。TIMESTAMP是以指定时间和UNIX epoch（ UNIX epoch时间为1970年1月1日00:00:00 ）之间的秒数差定义的。可以向TIMESTAMP隐性转换的数据类型有STRING（ 必须具有"yyyy-MM-dd HH:mm:SS[.ffffff]"格式。小数点后精度可选 ）。

CHAR

CHAR的长度是固定的，使用指定长度的固定长度表示字符串。DLI中实际存储为STRING类型。

VARCHAR

VARCHAR生成时会带有一个长度指定数，用来定义字符串中的最大字符数。如果一个向VARCHAR转换的STRING型中的字符个数超过了长度指定数，那么这个STRING会被

自动缩短。和STRING类型一样，VARCHAR末尾的空格数是有意义的，会影响比较结果。DLI中实际存储为STRING类型。

DATE

DATE类型只能和DATE、TIMESTAMP和STRING进行显式转换（cast），具体如**表1-67**所示。

表 1-67 cast 函数转换

显式转换	转换结果
cast(date as date)	相同DATE值。
cast(timestamp as date)	根据本地时区从TIMESTAMP得出年/月/日，将其作为DATE值返回。
cast(string as date)	如果字符串的形式是“yyyy-MM-dd”，将对应年/月/日作为DATE值返回。如果字符串不具有这种形式，返回空。
cast(date as timestamp)	根据本地时区生成并返回对应DATE的年/月/日零点的TIMESTAMP值。
cast(date as string)	根据DATE的年/月/日值生成并返回“yyyy-MM-dd”格式的字符串。

1.27.3 复杂数据类型

Spark SQL支持复杂数据类型，如**表1-68**所示。

表 1-68 复杂数据类型

数据类型	描述	使用格式
ARRAY	一组有序字段，使用指定的值构造ARRAY数组。可以为任意类型，要求所有字段的数据类型必须相同。	array(<value>,<value>[, ...]) 具体使用示例详见：**ARRAY示例**。
MAP	一组无序的键/值对，使用给定的Key和Value对生成MAP。键的类型必须是原生数据类型，值的类型可以是原生数据类型或复杂数据类型。同一个MAP键的类型必须相同，值的类型也必须相同。	map(K <key1>, V <value1>, K <key2>, V <value2>[, ...]) 具体使用示例详见：**MAP示例**。
STRUCT	一组命名的字段，字段的数据类型可以不同。	struct(<value1>,<value2>[, ...]) 具体使用示例详见：**STRUCT示例**。

使用限制

- 创建含有复杂数据类型字段的表时，该表存储格式不支持CSV（txt）。
- 如果表中含有复杂数据类型字段时，该表不支持CSV（txt）格式的文件数据导入。
- MAP数据类型建表必须指定schema，且不支持date、short、timestamp数据类型。
- 对于JSON格式OBS表，MAP的键类型只支持STRING类型。
- 由于MAP类型的键不能为NULL，MAP键不支持对插入数据进行可能出现NULL值类型之间的隐式转换，如：STRING类型转换为其他原生类型、FLOAT类型转换为TIMESTAMP类型、其他原生类型转换为DECIMAL类型等。
- STRUCT数据类型不支持double，boolean数据类型。

ARRAY 示例

创建表“array_test”，将“id”参数定义为“ARRAY<INT>”数据类型，“name”参数定义为“STRING”数据类型。建表成功后插入测试数据到“array_test”中。操作如下：

1. 创建表。

   CREATE TABLE array_test(name STRING, id ARRAY < INT >) USING PARQUET;

2. 插入测试数据。

   INSERT INTO array_test VALUES ('test',array(1,2,3,4));

   INSERT INTO array_test VALUES ('test2',array(4,5,6,7))

   INSERT INTO array_test VALUES ('test3',array(7,8,9,0));

3. 查询结果。

   查“array_test”表中的所有数据：

   SELECT * FROM array_test;

   ```
   test3    [7,8,9,0]
   test2    [4,5,6,7]
   test    [1,2,3,4]
   ```

   查“array_test”表中id数组第0个元素的数据。

   SELECT id[0] FROM array_test;

   ```
   7
   4
   1
   ```

MAP 示例

创建表“map_test”，将“score”参数定义为“map<STRING,INT>)”数据类型（键为STRING类型，值为INT类型）。建表成功后插入测试数据至“map_test”中。操作如下：

1. 创建表。

   CREATE TABLE map_test(id STRING, score map<STRING,INT>) USING PARQUET;

2. 插入测试数据。

   INSERT INTO map_test VALUES ('test4',map('math',70,'chemistry',84));

INSERT INTO map_test VALUES ('test5',map('math',85,'chemistry',97));

INSERT INTO map_test VALUES ('test6',map('math',88,'chemistry',80));

3. 查询结果。

   查询“map_test”表里的所有数据。

   SELECT * FROM map_test;

```
test6    {"chemistry":80,"math":88}
test5    {"chemistry":97,"math":85}
test4    {"chemistry":84,"math":70}
```

   查询“map_test”表中的数学成绩。

   SELECT id, score['Math'] FROM map_test;

```
test6    88
test5    85
test4    70
```

STRUCT 示例

创建表“struct_test”，将info定义为“STRUCT<name:STRING, age:INT>”数据类型（由name和age构成的字段，其中name为STRING类型，age为INT类型）。建表成功后插入测试数据至“struct_test”表中。操作如下：

1. 创建表。

   CREATE TABLE struct_test(id INT, info STRUCT<name:STRING,age:INT>) USING PARQUET;

2. 插入测试数据。

   INSERT INTO struct_test VALUES (8, struct('user1',23));

   INSERT INTO struct_test VALUES (9, struct('user2,25));

   INSERT INTO struct_test VALUES (10, struct('user3',26));

3. 查询结果。

   查询“struct_test”表中的所有数据。

   SELECT * FROM struct_test;

```
8{"name":"user1,"age":23}
10{"name":"user2,"age":26}
9{"name":"user3,"age":25}
```

   查询“struct_test”表中的**name**和**age**数据。

   SELECT id,info.name,info.age FROM struct_test;

```
8       user1    23
10      user2    26
9     user3     25
```

1.28 自定义函数

1.28.1 创建函数

功能描述

DLI支持创建使用UDF和UDTF等自定义函数应用于Spark作业开发当中。

语法格式

```
CREATE FUNCTION [db_name.]function_name AS class_name
  [USING resource,...]

resource:
  : JAR file_uri
```

或

```
CREATE OR REPLACE FUNCTION [db_name.]function_name AS class_name
  [USING resource,...]

resource:
  : JAR file_uri
```

注意事项

- 如果在数据库中存在同名的函数，系统将会报错。
- 只支持Hive语法创建函数。
- 请注意避免该场景：如果创建的自定义函数F1指定类C1，程序包名JAR1，创建自定义函数F2也指定类C1，程序包JAR2，因为F2和F1使用相同的类名，导致功能相互冲突，影响作业执行。

关键字

- USING <resources>：需要加载的资源。可以是JAR、文件或者URI的列表。

示例

创建函数mergeBill。

```
CREATE FUNCTION mergeBill AS 'com.xxx.hiveudf.MergeBill'
  using jar 'obs://onlyci-7/udf/MergeBill.jar';
```

1.28.2 删除函数

功能描述

删除函数。

语法格式

```
DROP [TEMPORARY] FUNCTION [IF EXISTS] [db_name.] function_name;
```

关键字

- TEMPORARY：所删除的函数是否为临时函数。
- IF EXISTS：所删除的函数不存在时使用，可避免系统报错。

注意事项

- 删除一个已存在的函数。如果要删除的函数不存在，则系统报错。
- 只支持HIVE语法。

示例

删除函数mergeBill。

```
DROP FUNCTION mergeBill;
```

1.28.3 显示函数详情

功能描述

查看指定函数的相关信息。

语法格式

```
DESCRIBE FUNCTION [EXTENDED] [db_name.] function_name;
```

关键字

EXTENDED：显示扩展使用信息。

注意事项

返回已有函数的元数据（实现类和用法），如果函数不存在，则系统报错。

示例

查看函数mergeBill的相关信息。

```
DESCRIBE FUNCTION mergeBill;
```

1.28.4 显示所有函数

功能描述

查看当前工程下所有的函数。

语法格式

```
SHOW [USER|SYSTEM|ALL] FUNCTIONS ([LIKE] regex | [db_name.] function_name);
```

其中regex为正则表达式，可以参考如下**表1-69**参数样例。

表 1-69 regex 参数举例说明

regex表达式	匹配含义
'xpath*'	表示匹配所有xpath开头的函数名。 例如：SHOW FUNCTIONS LIKE 'xpath*' ; 表示可以匹配到：xpath、xpath_int、xpath_string等等xpath开头的函数。
'x[a-z]+'	表示匹配以x开头，后面是a到z范围的一个到多个字符的函数名。如可以匹配到：xpath、xtest等。

regex表达式	匹配含义
'x.*h'	匹配以x开头，h结尾，中间为一个或多个字符的函数名。如可以匹配到：xpath、xtesth等。

其他更多正则表达式的使用，可参考官网说明。

关键字

LIKE：此限定符仅为兼容性而使用，没有任何实际作用。

注意事项

显示与给定正则表达式或函数名匹配的函数。如果未提供正则表达式或名称，则显示所有函数。如果声明了USER或SYSTEM，那么将分别显示用户定义的Spark SQL函数和系统定义的Spark SQL函数。

示例

查看当前的所有函数。

```
SHOW FUNCTIONS;
```

1.29 内置函数

1.29.1 日期函数

1.29.1.1 日期函数概览

DLI所支持的日期函数如**表1-70**所示。

表 1-70 日期/时间函数

命令格式	返回值	功能简介
add_months(string start_date, int num_months)	STRING	返回start_date在num_months个月之后的date。
current_date()	DATE	返回当前日期，如2016-07-04。
current_timestamp()	TIMESTAMP	返回TIMESTAMP类型的时间戳。

命令格式	返回值	功能简介
date_add(string startdate, int days)	STRING或DATE	给定时间，在此基础上加上指定的时间段。
dateadd(string date, bigint delta, string datepart)	STRING或DATE	dateadd函数用于按照指定的单位datepart和幅度delta修改date的值。 date：必填。日期值，string类型。 使用的时间格式为yyyy-mm-dd hh:mi:ss，例如2021-08-28 00:00:00。 delta：必填。修改幅度，BIGINT类型。 datepart：必填。指定修改的单位，STRING类型常量。
date_sub(string startdate, int days)	STRING	给定时间，在此基础上减去指定的时间段。
date_format(string date, string format)	STRING	用于将date按照format指定的格式转换为字符串。
datediff(string date1, string date2)	BIGINT	datediff函数用于计算两个时间date1、date2的日期差值。
datediff1(string date1, string date2, string datepart)	BIGINT	datediff1函数用于计算两个时间date1、date2的差值，将差值以指定的时间单位datepart表示。
datepart（string date，string datepart）	BIGINT	取日期中符合指定时间单位的字段值。
datetrunc (string date, string datepart)	STRING	datetrunc函数用于计算将日期date按照datepart指定的时间单位进行截取后的日期值。 date：必填。格式为yyyy-mm-dd或yyyy-mm-dd hh:mi:ss。 datepart：必填。STRING类型常量。支持扩展的日期格式。
day(string date)、dayofmonth(string date)	INT	返回指定时间的日期。
from_unixtime(bigint unixtime)	STRING	将时间戳转换为时间格式，格式为“yyyy-MM-dd HH:mm:ss”或“yyyyMMddHHmmss.uuuuuu”。 例如：select FROM_UNIXTIME(1608135036,'yyyy-MM-dd HH:mm:ss')

命令格式	返回值	功能简介
from_utc_timestamp(string timestamp, string timezone)	TIMESTAMP	将UTC的时间戳转化为timezone所对应的时间戳。
getdate()	STRING	获取当前系统时间。
hour(string date)	INT	返回指定时间的小时，范围为0到23。
isdate(string date , string format)	BOOLEAN	date：必填。STRING类型。会隐式转换为STRING类型后参与运算。 format：必填。STRING类型常量，不支持日期扩展格式。如果format中出现多余的格式串，则只取第一个格式串对应的日期数值，其余的会被视为分隔符。例如isdate("1234-yyyy", "yyyy-yyyy")，会返回True。
last_day(string date)	DATE	返回date所在月份的最后一天，格式为yyyy-MM-dd，如2015-08-31。
lastday(string date)	STRING	lastday函数用于返回date所在月的最后一天，返回值STRING类型，格式为yyyy-mm-dd hh:mi:ss。
minute(string date)	INT	返回指定时间的分钟，范围为0到59。
month(string date)	INT	返回指定时间的月份，范围为1至12月。
months_between(string date1, string date2)	DOUBLE	返回date1与date2之间的月份差。
next_day(string start_date, string day_of_week)	DATE	返回start_date之后最接近day_of_week的日期，格式为yyyy-MM-dd，day_of_week表示一周内的星期（如Monday、FRIDAY）。
quarter(string date)	INT	返回该date/timestamp/string所在的季度，如quarter('2015-04-01')=2。
second(string date)	INT	返回指定时间的秒，范围为0到59。
to_char(string date, string format)	STRING	将日期按照指定格式转换为字符串。
to_date(string timestamp)	DATE	返回时间中的年月日，例如：to_date("1970-01-01 00:00:00") = "1970-01-01"

命令格式	返回值	功能简介
to_date1(string date, string format)	STRING	返回STRING类型，格式为yyyy-mm-dd hh:mi:ss。date或format值为NULL时，返回NULL。 将指定格式的字符串转换为日期值。
to_utc_timestamp(string timestamp, string timezone)	TIMESTAMP	将timezone所对应的时间戳转换为UTC的时间戳。
trunc(string date, string format)	DTAE	将date按照特定的格式进行清零操作，支持格式为MONTH/MON/MM, YEAR/YYYY/YY，如trunc('2015-03-17', 'MM') = 2015-03-01。
unix_timestamp(string timestamp, string pattern)	BIGINT	如果不带参数的调用，返回一个Unix时间戳（从“1970-01-01 00:00:00”到现在的秒数）为无符号整数。
weekday (string date)	INT	返回日期值是当前周的第几天。
weekofyear(string date)	INT	返回指定日期是一年中的第几周，范围为0到53。
year(string date)	INT	返回指定日期中的年份。

1.29.1.2 add_months

add_months函数用于计算日期值增加指定月数后的日期。即start_date在num_months个月之后的date。

命令格式

```
add_months(string start_date, int num_months)
```

参数说明

表 1-71 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE或STRING	代表起始日期。 支持以下格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
num_months	是	INT	代表需要增加月的数量。

返回值说明

返回开始日期startdate增加num_months个月后的日期，返回值格式为yyyy-mm-dd。

返回值date类型的日期值。

说明

- startdate非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- startdate为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- startdate值为NULL时，返回报错。
- num_months值为NULL时，返回NULL。

示例代码

返回2023-05-26。

```
select add_months('2023-02-26',3);
```

返回2023-05-14。

```
select add_months('2023-02-14 21:30:00',3);
```

返回NULL。

```
select add_months('20230815',3);
```

返回NULL。

```
select add_months('2023-08-15 20:00:00',null);
```

1.29.1.3 current_date

current_date函数用于返回当前日期值。返回值格式为yyyy-mm-dd。

相似函数：**getdate**，getdate函数用于返回当前系统时间。返回值格式为yyyy-mm-dd hh:mi:ss。

命令格式

```
current_date()
```

参数说明

无

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd

示例代码

返回2023-08-16。

```
select current_date();
```

1.29.1.4 current_timestamp

CURRENT_TIMESTAMP函数用于返回当前时间戳。

命令格式

```
current_timestamp()
```

参数说明

无

返回值说明

返回TIMESTAMP类型的时间戳。

示例代码

返回1692002816300。

```
select current_timestamp();
```

1.29.1.5 date_add

date_add函数用于计算按照days幅度递增startdate日期的天数。

如需要获取当前日期基础上指定变动幅度的日期，可结合**current_date**或**getdate**函数共同使用。

请注意date_add函数与**date_sub**函数逻辑反。

命令格式

```
date_add(string startdate, int days)
```

参数说明

表 1-72 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE 或 STRING	代表起始日期。 支持以下格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
days	是	BIGINT	代表需要增加天的数量。 • days大于0，则为增加天数。 • days小于0，则减去天数。 • days等于0，即加0天，日期不变。 • days值为NULL时，返回NULL。

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- startdate非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- startdate为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- startdate值为NULL时，返回NULL。
- days值为NULL时，返回NULL。

示例代码

返回2023-03-01。加1天，结果超出当年2月份的最后1天，实际值为下个月的第1天。

```
select date_add('2023-02-28 00:00:00', 1);
```

返回2023-02-27。减1天。

```
select date_add(date '2023-02-28', -1);
```

返回2023-03-20。

```
select date_add('2023-02-28 00:00:00', 20);
```

假设当前时间为2023-08-14 16:00:00，返回2023-08-13。

```
select date_add(getdate(),-1);
```

返回NULL。

```
select date_add('2023-02-28 00:00:00', null);
```

1.29.1.6 dateadd

dateadd函数用于按照指定的单位datepart和幅度delta修改date的值。

如需要获取当前日期基础上指定变动幅度的日期，可结合**current_date**或**getdate**函数共同使用。

命令格式

```
dateadd(string date, bigint delta, string datepart)
```

参数说明

表 1-73 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表起始日期。 支持以下格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

参数	是否必选	参数类型	说明
delta	是	BIGINT	代表修改幅度。
datepart	是	BIGINT	代表修改的时间单位。 参数datepart支持扩展的日期格式：年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • MM代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回STRING类型的日期值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- delta或datepart值为NULL时，返回NULL。

示例代码

返回2023-08-15 17:00:00。加1天。

```
select dateadd( '2023-08-14 17:00:00', 1, 'dd');
```

返回2025-04-14 17:00:00。加20个月，月份溢出，年份加1。

```
select dateadd('2023-08-14 17:00:00', 20, 'mm');
```

返回2023-09-14 17:00:00。

```
select dateadd('2023-08-14 17:00:00', 1, 'mm');
```

返回2023-09-14。

```
select dateadd('2023-08-14', 1, 'mm');
```

假设当前时间为2023-08-14 17:00:00，返回2023-08-13 17:00:00。

```
select dateadd(getdate(),-1,'dd');
```

返回NULL。

```
select dateadd(date '2023-08-14', 1, null);
```


1.29.1.7 date_sub

date_sub函数按照days幅度递减startdate日期的天数。

如需要获取当前日期基础上指定变动幅度的日期，可结合**current_date**或**getdate**函数共同使用。

请注意date_sub函数与**date_add**函数逻辑反。

命令格式

```
date_sub(string startdate, int days)
```

参数说明

表 1-74 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE 或 STRING	代表起始日期。 支持以下格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
days	是	BIGINT	代表需要减少的天数。 • days大于0，则为增加天数。 • days小于0，则减去天数。 • days等于0，即加0天，日期不变。 • days值为NULL时，返回NULL。

返回值说明

返回DATE类型的日期值。

说明

- startdate非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- startdate为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- format值为NULL时，返回NULL。

示例代码

返回2023-08-12。减2天。

```
select date_sub('2023-08-14 17:00:00', 2);
```

返回2023-08-15。增1天。

```
select date_sub(date'2023-08-14', -1);
```

假设当前时间为2023-08-14 17:00:00，返回2023-08-13。

```
select date_sub(getdate(),1);
```

返回NULL。

```
select date_sub('2023-08-14 17:00:00', null);
```

1.29.1.8 date_format

date_format函数用于将date按照format指定的格式转换为字符串。

命令格式

```
date_format(string date, string format)
```

参数说明

表 1-75 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表待转换的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的格式。 格式为代表年月日时分秒的时间单位与任意字符的组合，其中： • yyyy代表年份。 • MM代表月份。 • dd代表天。 • HH代表24小时制时。 • hh代表12小时制时。 • mm代表分钟。 • ss代表秒。

返回值说明

按指定类型返回STRING类型的日期。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- format值为NULL时，返回NULL。

示例代码

返回2023-08-14。

```
select date_format('2023-08-14','yyyy-MM-dd');
```

返回2023-08。

```
select date_format('2023-08-14','yyyy-MM')
```

返回20230814。

```
select date_format('2023-08-14','yyyyMMdd')
```

1.29.1.9 datediff

datediff函数用于计算两个时间date1、date2的日期差值。

相似函数：**datediff1**，datediff1函数用于计算两个时间date1、date2的差值，将差值以指定的时间单位datepart表示。

命令格式

```
datediff(string date1, string date2)
```

参数说明

表 1-76 参数说明

参数	是否必选	参数类型	说明
date1	是	DATE 或 STRING	计算两个时间date1、date2的日期差值中的被减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
date2	是	DATE 或 STRING	计算两个时间date1、date2的日期差值的减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回BIGINT类型。

说明

- date1、date2非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date1、date2为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- 如果date1小于date2，返回值为负数。
- date1或date2值为NULL时，返回NULL。

示例代码

返回10。

```
select datediff('2023-06-30 00:00:00', '2023-06-20 00:00:00');
```

返回11。

```
select datediff(date '2023-05-21', date '2023-05-10');
```

返回NULL。

```
select datediff(date '2023-05-21', null);
```

1.29.1.10 datediff1

datediff1函数用于计算两个时间date1、date2的差值，将差值以指定的时间单位datepart表示。

相似函数：**datediff**，datediff函数用于计算两个时间date1、date2的日期差值，不支持指定返回的时间单位。

命令格式

```
datediff1(string date1, string date2, string datepart)
```

参数说明

表 1-77 参数说明

参数	是否必选	参数类型	说明
date1	是	DATE 或 STRING	计算两个时间date1、date2的日期差值中的被减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

参数	是否必选	参数类型	说明
date2	是	DATE 或 STRING	计算两个时间date1、date2的日期差值的减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
datepart	是	STRING	代表需要返回的时间单位。 参数datepart支持扩展的日期格式：年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • MM代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BIGINT类型。

说明

- date1、date2非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- date1、date2为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- 如果date1小于date2，返回值为负数。
- date1或date2值为NULL时，返回NULL。
- datepart值为NULL时，返回NULL。

示例代码

返回14400。

```
select datediff1('2023-06-30 00:00:00', '2023-06-20 00:00:00', 'mi');
```

返回10。

```
select datediff1(date '2023-06-21', date '2023-06-11', 'dd');
```

返回NULL。

```
select datediff1(date '2023-05-21', date '2023-05-10', null);
```

返回NULL。

```
select datediff1(date '2023-05-21', null, 'dd');
```


1.29.1.11 datepart

datepart函数用于计算日期date中符合指定时间单位datepart的值。

命令格式

```
datepart ( string date，string datepart )
```

参数说明

表 1-78 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表起始日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
datepart	是	STRING	代表需要返回的时间单位。 参数datepart支持扩展的日期格式：年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • MM代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BIGINT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- datepart值为NULL时，返回NULL。
- datepart值为NULL时，返回NULL。

示例代码

返回2023。

```
select datepart(date '2023-08-14 17:00:00', 'yyyy');
```

返回2023。

```
select datepart('2023-08-14 17:00:00', 'yyyy');
```

返回59。

```
select datepart('2023-08-14 17:59:59', 'mi')
```

返回NULL。

```
select datepart(date '2023-08-14', null);
```

1.29.1.12 datetrunc

datetrunc函数用于计算将日期date按照datepart指定的时间单位进行截取后的日期值。

截取datepart之前的部分，除截取的部分外自动填充为默认值。可参考**示例代码**。

命令格式

```
datetrunc (string date, string datepart)
```

参数说明

表 1-79 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表起始日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
datepart	是	STRING	代表需要返回的时间单位。 参数datepart支持扩展的日期格式：年-year、月-month或-mon、日-day和小时-hour。 • yyyy代表年份。 • MM代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回DATE或STRING类型的日期值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- datepart值为NULL时，返回NULL。
- datepart值为时、分、秒时，按照天截取返回

示例代码

静态数据示例

返回2023-01-01 00:00:00。

```
select datetrunc('2023-08-14 17:00:00', 'yyyy');
```

返回2023-08-01 00:00:00。

```
select datetrunc('2023-08-14 17:00:00', 'month');
```

返回2023-08-14。

```
select datetrunc('2023-08-14 17:00:00', 'DD');
```

返回2023-01-01。

```
select datetrunc('2023-08-14', 'yyyy');
```

返回2023-08-14 17:00:00。

```
select datetrunc('2023-08-14 17:11:11', 'hh');
```

返回NULL。

```
select datetrunc('2023-08-14', null);
```

1.29.1.13 day/dayofmonth

day函数用于返回指定日期的天。

命令格式

```
day(string date)、dayofmonth(string date)
```

参数说明

表 1-80 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回1。

```
select day('2023-08-01');
```

返回NULL。

```
select day('20230816');
```

返回NULL。

```
select day(null);
```

1.29.1.14 from_unixtime

from_unixtime函数用于计算将数字型的UNIX值代表的时间戳转换为日期值。

命令格式

```
from_unixtime(bigint unixtime)
```

参数说明

表 1-81 参数说明

参数	是否必选	参数类型	说明
unixtime	是	BIGINT	UNIX格式的时间戳。代表需要转换的时间戳 此处参数应填正常UNIX格式时间戳前十位。

返回值说明

返回STRING类型的日期值，格式为yyyy-mm-dd hh:mi:ss。

说明

unixtime值为NULL时，返回NULL。

示例代码

返回2023-08-16 09:39:57。

```
select from_unixtime(1692149997);
```

返回NULL。

```
select from_unixtime(NULL);
```

表数据示例

```
select unixdate, from_unixtime(unixdate) as timestamp_from_unixtime from database_t;
输出：
+-----------------+-----------------------------+
| unixdate            | timestamp_from_unixtime   |
+-----------------+-----------------------------+
| 1690944759224  | 2023-08-02 10:52:39          |
| 1690944999811  | 2023-08-02 10:56:39          |
| 1690945005458  | 2023-08-02 10:56:45          |
| 1690945011542  | 2023-08-02 10:56:51          |
| 1690945023151  | 2023-08-02 10:57:03          |
+-----------------+-----------------------------+
```

1.29.1.15 from_utc_timestamp

from_utc_timestamp函数用于计算将UTC的时间戳转化为timezone所对应的UNIX格式的时间戳。

命令格式

```
from_utc_timestamp(string timestamp, string timezone)
```

参数说明

表 1-82 参数说明

参数	是否必选	参数类型	说明
timestamp	是	DATE STRING TINYINT SMALLINT INT BIGINT	代表待转换的时间。 DATE或STRING类型的日期值，或TINYINT、SMALLINT、INT或BIGINT类型的时间戳。 格式： yyyy-mm-dd yyyy-mm-dd hh:mi:ss yyyy-mm-dd hh:mi:ss.ff3
timezone	是	STRING	代表需要转换的目标时区。

返回值说明

返回TIMESTAMP类型的时间戳。

说明

- timestamp非DATE或STRING类型时，返回报错，错误信息：data type mismatch；
- timestamp为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL；
- timestamp值为NULL时，返回NULL。
- timezone值为NULL时，返回NULL。

示例代码

返回1691978400000（代表2023-08-14 10:00:00）。

```
select from_utc_timestamp('2023-08-14 17:00:00','PST');
```

返回1691917200000（代表2023-08-13 17:00:00）。

```
select from_utc_timestamp(date '2023-08-14 00:00:00','PST');
```

返回NULL。

```
select from_utc_timestamp('2023-08-13',null);
```

1.29.1.16 getdate

getdate函数用于返回当前系统时间。返回值格式为yyyy-mm-dd hh:mi:ss。

相似函数：**current_date**，current_date函数用于返回当前日期值。返回值格式为yyyy-mm-dd。

命令格式

```
getdate()
```

参数说明

无

返回值说明

返回STRING类型的日期值，格式为yyyy-mm-dd hh:mi:ss。

示例代码

假设当前时间为2023-08-10 10:54:00，返回2023-08-10 10:54:00。

```
select getdate();
```

1.29.1.17 hour

hour函数用于返回指定时间的小时，范围为0到23。

命令格式

```
hour(string date)
```


参数说明

表 1-83 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回10。

```
select hour('2023-08-10 10:54:00');
```

返回12。

```
select hour('12:00:00');
```

返回NULL。

```
select hour('20230810105600');
```

返回NULL。

```
select hour(null);
```

1.29.1.18 isdate

isdate函数用于判断一个日期字符串能否根据指定的格式转换为一个日期值。

命令格式

```
isdate(string date , string format)
```

参数说明

表 1-84 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要判断的字符串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，会隐式转换为STRING类型后参与运算 格式为任意字符串。
format	是	STRING	代表需要转换的目标日期格式。 STRING类型常量，不支持日期扩展格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： • yyyy代表年份。 • mm代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BOOLEAN类型的值。

说明

date或format值为NULL时，返回NULL。

示例代码

返回true。

```
select isdate('2023-08-10','yyyy-mm-dd');
```

返回false。

```
select isdate(123456789,'yyyy-mm-dd');
```

1.29.1.19 last_day

last_day函数用于返回date所在月份的最后一天。

相似函数：**lastday**，lastday函数用于返回date所在月的最后一天，截取到天，时分秒部分为00:00:00。

命令格式

```
last_day(string date)
```

参数说明

表 1-85 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回2023-08-31。

```
select last_day('2023-08-15');
```

返回2023-08-31。

```
select last_day('2023-08-10 10:54:00');
```

返回NULL。

```
select last_day('20230810');
```

1.29.1.20 lastday

lastday函数用于返回date所在月的最后一天，截取到天，时分秒部分为00:00:00。

相似函数：**last_day**，last_day函数用于返回date所在月份的最后一天。返回值格式为：yyyy-mm-dd。

命令格式

```
lastday(string date)
```

参数说明

表 1-86 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回STRING类型的日期值。格式为yyyy-mm-dd hh:mi:ss。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回2023-08-31。

```
select lastday('2023-08-10');
```

返回2023-08-31 00:00:00。

```
select lastday ('2023-08-10 10:54:00');
```

返回NULL。

```
select lastday (null);
```

1.29.1.21 minute

minute函数用于返回指定时间的分钟，范围为0到59。

命令格式

```
minute(string date)
```

参数说明

表 1-87 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回54。

```
select minute('2023-08-10 10:54:00');
```

返回54。

```
select minute('10:54:00');
```

返回NULL。

```
select minute('20230810105400');
```

返回NULL。

```
select minute(null);
```

1.29.1.22 month

month函数用于返回指定时间的月份，范围为1至12月。

命令格式

```
month(string date)
```

参数说明

表 1-88 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 date取值为STRING类型格式时，至少要包含yyyy-mm-dd且不含多余的字符串。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回8。

```
select month('2023-08-10 10:54:00');
```

返回NULL。

```
select month('20230810');
```

返回NULL。

```
select month(null);
```

1.29.1.23 months_between

months_between函数用于返回date1与date2之间的月份差。

命令格式

```
months_between(string date1, string date2)
```

参数说明

表 1-89 参数说明

参数	是否必选	参数类型	说明
date1	是	DATE 或 STRING	代表被减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
date2	是	DATE 或 STRING	代表减数。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回DOUBLE类型的值。

说明

- date1、date2非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date1、date2为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- 当date1晚于date2时，返回值为正。当date2晚于date1时，返回值为负。
- 当date1和date2分别对应两个月的最后一天，返回整数月；否则计算方式为date1减去date2的天数除以31天。
- date1或date2值为NULL时，返回NULL。

示例代码

返回0.0563172。

```
select months_between('2023-08-16 10:54:00', '2023-08-14 17:00:00');
```

返回0.06451613。

```
select months_between('2023-08-16','2023-08-14');
```

返回NULL。

```
select months_between('2023-08-16',null);
```

1.29.1.24 next_day

next_day函数用于返回start_date之后最接近day_of_week的日期。

命令格式

```
next_day(string start_date, string day_of_week)
```

参数说明

表 1-90 参数说明

参数	是否必选	参数类型	说明
start_date	是	DATE 或 STRING	代表需要处理的日期。 start_date取值为STRING类型格式时，至少要包含yyyy-mm-dd且不含多余的字符串。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
day_of_week	是	STRING	一个星期前2个或3个字母，或者一个星期的全名。例如TU代表星期二。

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- start_date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- start_date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- start_date值为NULL时，返回NULL。
- day_of_week值为NULL时，返回NULL。

示例代码

返回2023-08-22。

```
select next_day('2023-08-16','TU');
```

返回2023-08-22。

```
select next_day('2023-08-16 10:54:00','TU');
```

返回2023-08-23。

```
select next_day('2023-08-16 10:54:00','WE');
```

返回NULL。

```
select next_day('20230816','TU');
```

返回NULL。

```
select next_day('20230816 20:00:00',null);
```

1.29.1.25 quarter

quarter函数用于返回该date所在的季度，范围为1~4。

命令格式

```
quarter(string date)
```

参数说明

表 1-91 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回3。

```
select quarter('2023-08-16 10:54:00');
```

返回3。

```
select quarter('2023-08-16');
```

返回NULL。

```
select quarter(null);
```

1.29.1.26 second

second函数用于返回指定时间的秒，范围为0到59。

命令格式

```
second(string date)
```

参数说明

表 1-92 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回36。

```
select second('2023-08-16 10:54:36');
```

返回36。

```
select second('10:54:36');
```

返回NULL。

```
 select second('20230816105436');
```

返回NULL。

```
select second(null);
```

1.29.1.27 to_char

to_char函数用于将日期按照指定格式转换为字符串。

命令格式

```
to_char(string date, string format)
```

参数说明

表 1-93 参数说明

参数	是否必选	参数类型	说明
date	是	DATE 或 STRING	代表需要处理的日期。 格式为： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的目标日期格式。 STRING类型常量，不支持日期扩展格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： ● yyyy代表年份。 ● mm代表月份。 ● dd代表天。 ● hh代表小时。 ● mi代表分钟。 ● ss代表秒。

返回值说明

返回STRING类型。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- format值为NULL时，返回NULL。

示例代码

返回静态数据示例2023-08*16。

```
select to_char('2023-08-16 10:54:36', '静态数据示例yyyy-mm*dd');
```

返回20230816。

```
select to_char('2023-08-16 10:54:36', 'yyyymmdd');
```

返回NULL。

```
select to_char('静态数据示例2023-08-16', '静态数据示例yyyy-mm*dd');
```

返回NULL。

```
select to_char('20230816', 'yyyy');
```


返回NULL。

```
select to_char('2023-08-16 10:54:36', null);
```

1.29.1.28 to_date

to_date函数用于返回时间中的年月日。

相似函数：**to_date1**，to_date1函数用于将指定格式的字符串转换为日期值，支持指定转换的日期格式。

命令格式

```
to_date(string timestamp)
```

参数说明

表 1-94 参数说明

参数	是否必选	参数类型	说明
timestamp	是	DATE STRING	代表待处理的时间。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- timestamp非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- timestamp为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。

示例代码

返回2023-08-16。

```
select to_date('2023-08-16 10:54:36');
```

返回NULL。

```
select to_date(null);
```

1.29.1.29 to_date1

to_date1函数用于将指定格式的字符串转换为日期值。

相似函数：**to_date**，to_date函数用于返回时间中的年月日，不支持指定转换的日期格式。

命令格式

```
to_date1(string date, string format)
```

参数说明

表 1-95 参数说明

参数	是否必选	参数类型	说明
date	是	STRING	要转换的字符串。 格式： ● yyyy-mm-dd ● yyyy-mm-dd hh:mi:ss ● yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的日期格式。 STRING类型常量，不支持日期扩展格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： ● yyyy代表年份。 ● mm代表月份。 ● dd代表天。 ● hh代表小时。 ● mi代表分钟。 ● ss代表秒。

返回值说明

返回STRING类型的日期值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- format值为NULL时，返回yyyy-mm-dd格式的日期值。

示例代码

返回2023-08-16 10:54:36

```
select to_date1('2023-08-16 10:54:36','yyyy-mm-dd hh:mi:ss');
```

返回2023-08-16 00:00:00。

```
select to_date1('2023-08-16','yyyy-mm-dd');
```

返回NULL。

```
select to_date1(null);
```

返回2023-08-16。

```
select to_date1('2023-08-16 10:54:36');
```

1.29.1.30 to_utc_timestamp

to_utc_timestamp函数用于将timezone所对应的时间戳转换为UTC的时间戳。

命令格式

```
to_utc_timestamp(string timestamp, string timezone)
```

参数说明

表 1-96 参数说明

参数	是否必选	参数类型	说明
timestamp	是	DATE STRING TINYINT SMALLINT INT BIGINT	代表待处理的时间。 DATE或STRING类型的日期值，或TINYINT、SMALLINT、INT或BIGINT类型的时间戳。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
timezone	是	STRING	代表需要转换的目标时区。

返回值说明

返回BIGINT类型值。

说明

- timestamp非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- timestamp为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- timestamp值为NULL时，返回NULL。
- timezone值为NULL时，返回NULL。

示例代码

返回1692028800000。

```
select to_utc_timestamp('2023-08-14 17:00:00','PST');
```

返回null。

```
select to_utc_timestamp(null);
```

1.29.1.31 trunc

trunc函数用于将date按照特定的格式进行清零操作。

清零操作即返回默认值，年、月、日的默认值为01，时、分、秒、毫秒 的默认值为00。

命令格式

```
trunc(string date, string format)
```

参数说明

表 1-97 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
format	是	STRING	代表需要转换的目标日期格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： • yyyy代表年份。 • MM代表月份。

返回值说明

返回DATE类型的日期值，格式为yyyy-mm-dd。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。
- format值为NULL时，返回NULL。

示例代码

返回2023-08-01。

```
select trunc('2023-08-16', 'MM');
```

返回2023-08-01。

```
select trunc('2023-08-16 10:54:36', 'MM');
```

返回NULL。

```
select trunc(null, 'MM');
```

1.29.1.32 unix_timestamp

unix_timestamp函数用于将日期值转化为数字型的UNIX格式的日期值。

函数返回值将返回正常UNIX格式时间戳前十位。

命令格式

```
unix_timestamp(string timestamp, string pattern)
```

参数说明

表 1-98 参数说明

参数	是否必选	参数类型	说明
timestamp	否	DATE或STRING	代表待转换的日期值。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3
pattern	否	STRING	代表需要转换的格式。 pattern为空时，默认为yyyy-MM-dd hh:mm:ss格式。 format:格式为代表年月日时分秒的时间单位与任意字符的组合，其中： • yyyy代表年份。 • MM代表月份。 • dd代表天。 • hh代表小时。 • mi代表分钟。 • ss代表秒。

返回值说明

返回BIGINT类型的值。

说明

- timestamp值为NULL时，返回NULL。
- timestamp和pattern都为空时，返回从“1970-01-01 00:00:00”到现在的秒数代表的时间戳。

示例代码

返回1692149997。


```
select unix_timestamp('2023-08-16 09:39:57')
```

假设当前系统时间为2023-08-16 10:23:16，返回1692152596。

```
select unix_timestamp();
```

返回1692115200（即2023-08-16 00:00:00）。

```
select unix_timestamp("2023-08-16 10:56:45", "yyyy-MM-dd");
```

表数据示例

```
select timestamp1, unix_timestamp(timestamp1) as date1_unix_timestamp, timestamp2,
unix_timestamp(datetime1) as date2_unix_timestamp, timestamp3, unix_timestamp(timestamp1) as
date3_unix_timestamp from database_t;输出：
+------------+------------------------+----------------------+---------------------- --
+-----------------------------------+---------------------------+
| timestamp1| date1_unix_timestamp | timestamp2              | date2_unix_timestamp |
timestamp3                          | date3_unix_timestamp      |
+------------+------------------------+----------------------+------------------------
+-----------------------------------+---------------------------+
| 2023-08-02 | 1690905600000          | 2023-08-02 11:09:14 | 1690945754793          | 2023-01-11
00:00:00.123456789 | 1673366400000             |
| 2023-08-03 | 1690992000000          | 2023-08-02 11:09:31 | 1690945771994          | 2023-02-11
00:00:00.123456789 | 1676044800000             |
| 2023-08-04 | 1691078400000          | 2023-08-02 11:09:41 | 1690945781270          | 2023-03-11
00:00:00.123456789 | 1678464000000             |
| 2023-08-05 | 1691164800000          | 2023-08-02 11:09:48 | 1690945788874          | 2023-04-11
00:00:00.123456789 | 1681142400000             |
| 2023-08-06 | 1691251200000          | 2023-08-02 11:09:59 | 1690945799099          | 2023-05-11
00:00:00.123456789 | 1683734400000             |
+------------+------------------------+----------------------+------------------------
+-----------------------------------+---------------------------+
```

1.29.1.33 weekday

weekday函数用于返回日期值是当前周的第几天。

命令格式

```
weekday (string date)
```

参数说明

表 1-99 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- 周一作为一周的第一天，返回值为0。其他日期依次递增，周日返回6。
- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回2。

```
select weekday ('2023-08-16 10:54:36');
```

返回NULL。

```
select weekday (null);
```

1.29.1.34 weekofyear

weekofyear函数用于返回指定日期是一年中的第几周，范围为0到53。

命令格式

```
weekofyear(string date)
```

参数说明

表 1-100 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回33。

```
select weekofyear('2023-08-16 10:54:36');
```

返回NULL。

```
select weekofyear('20230816');
```

返回NULL。

```
select weekofyear(null);
```

1.29.1.35 year

year函数用于返回指定日期中的年份。

命令格式

```
year(string date)
```

参数说明

表 1-101 参数说明

参数	是否必选	参数类型	说明
date	是	DATE或STRING	需要处理的日期。 格式： • yyyy-mm-dd • yyyy-mm-dd hh:mi:ss • yyyy-mm-dd hh:mi:ss.ff3

返回值说明

返回INT类型的值。

说明

- date非DATE或STRING类型时，返回报错，错误信息：data type mismatch。
- date为DATE或STRING类型，但不符合日期值的入参格式时，返回NULL。
- date值为NULL时，返回NULL。

示例代码

返回2023。

```
select year('2023-08-16 10:54:36');
```

返回NULL。

```
select year('23-01-01');
```

返回NULL。

```
select year('2023/08/16');
```

返回NULL。

```
select year(null);
```

1.29.2 字符串函数

1.29.2.1 字符串函数概览

DLI所支持的字符函数如**字符串函数**所示。

表 1-102 字符串函数

命令格式	返回值	功能简介
ascii(string <str>)	BIGINT	返回字符串中首字符的数字值。
concat(array<T> <a>, array<T> <b>[,...]), concat(string <str1>, string <str2>[,...])	ARRAY或STRING	连接多个字符串，合并为一个字符串，可以接受任意数量的输入字符串。
concat_ws(string <separator>, string <str1>, string <str2>[,...]), concat_ws(string <separator>, array<string> <a>)	ARRAY或STRUCT	连接多个字符串，字符串之间以指定的分隔符分隔。
char_matchcount(string <str1>, string <str2>)	BIGINT	计算str1中有多少个字符出现在str2中。
encode(string <str>, string <charset>)	BINARY	将str按照charset格式进行编码。
find_in_set(string <str1>, string <str2>)	BIGINT	查找字符串str1在以逗号（,）分隔的字符串str2中的位置，从1开始计数。
get_json_object(string <json>, string <path>)	STRING	根据所给路径对json对象进行解析，当json对象非法时将返回NULL。
instr(string <str>, string <substr>)	INT	返回substr在str中最早出现的下标。当参数中出现NULL时，返回NULL，当str中不存在substr时返回0，注意下标从1开始。
instr1(string <str1>, string <str2>[, bigint <start_position>[, bigint <nth_appearance>]])	BIGINT	instr1函数用于计算子串str2在字符串str1中的位置。
initcap(string A)	STRING	将文本字符串转换成首字母大写其余字母小写的形式。
keyvalue(string <str>, [string <split1>,string <split2>,] string <key>)	STRING	用于计算将字符串str按照split1进行切分，并按split2将每组变成Key-Value对，返回key所对应的Value。

命令格式	返回值	功能简介
length(string <str>)	BIGINT	返回字符串的长度。
lengthb(string <str>)	STRING	计算字符串str以字节为单位的长度。
levenshtein(string A, string B)	INT	返回两个字符串之间的Levenshtein距离，如levenshtein('kitten','sitting') =3。
locate(string <substr>, string <str>[, bigint <start_pos>])	BIGINT	用于在str中查找substr的位置。
lower(string A) ，lcase(string A)	STRING	将文本字符串转换成字母全部小写的形式。
lpad(string <str1>, int <length>, string <str2>)	STRING	用于返回指定长度的字符串，给定字符串str1长度小于指定长度length时，由指定字符str2从左侧填补。
ltrim([<trimChars>,] string <str>)	STRING	删除字符串左边的空格，其他的空格保留。
parse_url(string urlString, string partToExtract [, string keyToExtract])	STRING	返回给定URL的指定部分，partToExtract的有效值包括HOST，PATH， QUERY， REF，PROTOCOL， AUTHORITY，FILE和USERINFO。 例如：parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') 返回'facebook.com'.。 当第二个参数为QUERY时，可以使用第三个参数提取特定参数的值，例如：parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') 返回'v1'。
printf(String format, Obj... args)	STRING	将输入按特定格式打印输出。
regexp_count(string <source>, string <pattern>[, bigint <start_position>])	BIGINT	用于计算source中从start_position位置开始，匹配指定pattern的子串数。
regexp_extract(string <source>, string <pattern>[, bigint <groupid>])	STRING	用于将字符串source按照pattern的分组规则进行字符串匹配，返回第groupid个组匹配到的字符串内容。
replace(string <str>, string <old>, string <new>)	STRING	将字符串中与指定字符串匹配的子串替换为另一字符串。

命令格式	返回值	功能简介
• 适用于Spark2.4.5：regexp_replace(string <source>, string <pattern>, string <replace_string>) • 适用于Spark3.3.1：regexp_replace(string <source>, string <pattern>, string <replace_string>[, bigint <occurrence>])	STRING	• 适用于Spark2.4.5：用于将source字符串中第occurrence次匹配pattern的子串，以及之后匹配pattern的子串，全都替换成指定字符串replace_string后，返回结果字符 • 适用于Spark3.3.1：用于将source字符串中第occurrence次匹配pattern的子串，以及之后匹配pattern的子串，全都替换成指定字符串replace_string后，返回结果字符
regexp_replace1(string <source>, string <pattern>, string <replace_string>[, bigint <occurrence>])	STRING	将source字符串中第occurrence次匹配pattern的子串，替换成指定字符串replace_string后，返回结果字符串。
regexp_instr(string <source>, string <pattern>[,bigint <start_position>[, bigint <occurrence>[, bigint <return_option>]]])	BIGINT	用于计算字符串source从start_position开始，与pattern第occurrence次匹配的子串的起始或结束位置。
regexp_substr(string <source>, string <pattern>[, bigint <start_position>[, bigint <occurrence>]])	STRING	用于计算从start_position位置开始，source中第occurrence次匹配指定pattern的子串。
repeat(string <str>, bigint <n>)	STRING	重复N次字符串。
reverse(string <str>)	STRING	返回倒序字符串。
rpad(string <str1>, int <length>, string <str2>)	STRING	用于将字符串str2将字符串str1向右补足到length位。
rtrim([<trimChars>,]string <str>)， rtrim(trailing [<trimChars>] from <str>)	STRING	删除字符串右边的空格，其他的空格保留。
soundex(string <str>)	STRING	从str返回一个soundex字符串，如soundex('Miller')= M460。
space(bigint <n>)	STRING	返回指定数量的空格。

命令格式	返回值	功能简介
substr(string <str>, bigint <start_position>[, bigint <length>]) ，substring(string <str>, bigint <start_position>[, bigint <length>])	STRING	用于返回字符串str从start_position开始，长度为length的子串。
substring_index(string <str>, string <separator>, int <count>)	STRING	用于截取字符串str第count个分隔符之前的字符串。如果count为正，则从左边开始截取。如果count为负，则从右边开始截取。
split_part(string <str>, string <separator>, bigint <start>[, bigint <end>])	STRING	用于依照分隔符separator拆分字符串str，返回从start部分到end部分的子串（闭区间）。
translate(string|char|varchar input, string|char|varchar from, string|char|varchar to)	STRING	将input字符串中的所出现的字符或者字符串from用字符或者字符串to替换。例如：将abcde中的bcd替换成BCD，translate("abcde", "bcd", "BCD")
trim([<trimChars>,]string <str>)， trim([BOTH] [<trimChars>] from <str>)	STRING	删除字符串两端的空格，字符之间的空格保留。
upper(string A) ，ucase(string A)	STRING	将文本字符串转换成字母全部大写的形式。

1.29.2.2 ascii

ascii函数用于返回字符串str第一个字符的ASCII码。

命令格式

```
ascii(string <str>)
```

参数说明

表 1-103 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会自动转换为STRING类型后参与运算。 例如“ABC”

返回值说明

返回BIGINT的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

- 返回字符串ABC第一个字符的ASCII码。命令示例如下。

  返回97。

  ```
  select ascii('ABC');
  ```

- 输入参数为NULL。命令示例如下。

  返回NULL。

  ```
  select ascii(null);
  ```

1.29.2.3 concat

concat函数用于拼接数组或字符串。

命令格式

输入为ARRAY数组：将多个ARRAY数组中的所有元素连接在一起，生成一个新的ARRAY数组。

```
concat(array<T> <a>, array<T> <b>[,...])
```

输入为字符串：将多个字符串连接在一起，生成一个新的字符串。

```
concat(string <str1>, string <str2>[,...])
```

参数说明

- 输入为ARRAY数组

  表 1-104 参数说明

参数	是否必选	参数类型	说明
a、b	是	STRING	ARRAY数组。 array<T>中的T指代ARRAY数组元素的数据类型，数组中的元素可以为任意类型。 a和b中元素的数据类型必须一致。数组中的元素为NULL值时会参与运算。

- 输入为字符串

表 1-105 参数说明

参数	是否必选	参数类型	说明
str1、str2	是	STRING	字符串。 如果输入参数为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会自动转换为STRING类型后参与运算，其他类型会返回报错。

返回值说明

返回ARRAY数组或STRING的值。

说明

- 返回ARRAY类型。如果任一输入ARRAY数组为NULL，返回结果为NULL。
- 返回STRING类型。如果没有参数或任一参数为NULL，返回结果为NULL。

示例代码

- 连接ARRAY数组array(1, 2)和array(2, -2)。命令示例如下。

  返回[1, 2, 2, -2]。

  ```
  select concat(array(1, 2), array(2, -2));
  ```

- 任一ARRAY数组为NULL。命令示例如下。

  返回NULL。

  ```
  select concat(array(10, 20), null);
  ```

- 连接字符串ABC和DEF。命令示例如下。

  返回ABCDEF。

  ```
  select concat('ABC','DEF');
  ```

- 输入为空。命令示例如下。

  返回NULL。

  ```
  select concat();
  ```

- 任一字符串输入为NULL。命令示例如下。

  返回NULL。

  ```
  select concat('abc', 'def', null);
  ```

1.29.2.4 concat_ws

concat_ws函数用于连接多个字符串，字符串之间以指定的分隔符分隔。

命令格式

```
concat_ws(string <separator>, string <str1>, string <str2>[,...])
```

或

```
concat_ws(string <separator>, array<string> <a>)
```

返回将参数中的所有字符串或ARRAY数组中的元素按照指定的分隔符连接在一起的结果。

参数说明

表 1-106 参数说明

参数	是否必选	参数类型	说明
separator	是	STRING	STRING类型的分隔符。
str1、str2	是	STRING	至少要指定2个字符串。 STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
a	是	ARRAY	数组中的元素为STRING类型。

返回值说明

返回STRING类型或STRUCT类型的值。

说明

- str1或str2非STRING、BIGINT、DECIMAL、DOUBLE或DATETIME类型时，返回报错。
- 如果参数（待拼接字符）为NULL，则会忽略这个参数
- 如果没有输入参数（待拼接字符）返回NULL。

示例代码

- 将字符串ABC和DEF通过:连接。命令示例如下。

  返回ABC:DEF。

  ```
  select concat_ws(':','ABC','DEF');
  ```

- 任一输入参数为NULL。命令示例如下。

  返回avg:18。

  ```
  select concat_ws(':','avg',null,'18');
  ```

- 将ARRAY数组array('name', 'lilei')中的元素通过:连接。命令示例如下。

  返回name:lilei。

  ```
  select concat_ws(':',array('name', 'lilei'));
  ```

1.29.2.5 char_matchcount

char_matchcount函数用于计算str1中有多少个字符出现在str2中。

命令格式

```
char_matchcount(string <str1>, string <str2>)
```

参数说明

表 1-107 参数说明

参数	是否必选	参数类型	说明
str1、str2	是	STRING	待计算的字符串str1、str2。

返回值说明

返回BIGINT类型。

说明

str1或str2值为NULL时，返回NULL。

示例代码

返回3。

```
select char_matchcount('abcz','abcde');
```

返回NULL。

```
select char_matchcount(null,'abcde');
```

1.29.2.6 encode

encode函数用于使用charset的编码方式对str进行编码。

命令格式

encode(string <str>, string <charset>)

参数说明

表 1-108 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	至少要指定2个字符串。 STRING类型。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
charset	是	STRING	编码格式。 取值范围为：UTF-8、UTF-16、UTF-16LE、UTF-16BE、ISO-8859-1、US-ASCII。

返回值说明

返回BINARY类型的值。

说明

str或charset值为NULL时，返回NULL。

示例代码

- 将字符串abc按照UTF-8格式编码。命令示例如下。

  返回abc。

```
select encode("abc", "UTF-8");
```

- 任一输入参数为NULL。命令示例如下。

  返回结果为NULL。

```
select encode("abc", null);
```

1.29.2.7 find_in_set

find_in_set函数用于查找字符串str1在以逗号（,）分隔的字符串str2中的位置，从1开始计数。

命令格式

```
find_in_set(string <str1>, string <str2>)
```

参数说明

表 1-109 参数说明

参数	是否必选	参数类型	说明
str1	是	STRING	待查找的字符串。
str2	是	STRING	以逗号（,）分隔的字符串。

返回值说明

返回BIGINT类型的值。

说明

- 当str2中无法匹配到str1或str1中包含逗号（,）时，返回0。
- 当str1或str2值为NULL时，返回NULL。

示例代码

- 查找字符串ab在字符串abc,123,ab,c中的位置。命令示例如下。

  返回3。

```
select find_in_set('ab', 'abc,123,ab,c');
```

- 查找字符串hi在字符串abc,123,ab,c中的位置。命令示例如下。

返回0。

```
select find_in_set('hi', 'abc,123,ab,c');
```

- 任一输入参数为NULL。命令示例如下。

  返回NULL。

  ```
  select find_in_set(null, 'abc,123,ab,c');
  ```

1.29.2.8 get_json_object

get_json_object函数用于根据所给路径对json对象进行解析，当json对象非法时将返回NULL。

命令格式

```
get_json_object(string <json>, string <path>)
```

参数说明

表 1-110 参数说明

参数	是否必选	参数类型	说明
json	是	STRING	标准的JSON格式对象，格式为{Key:Value,Key:Value,...}
path	是	STRING	表示在json中的path，以$开头。不同字符的含义如下： • $表示根节点。 • .表示子节点。 • []表示[number]表示数组下标，从0开始。 • *表示Wildcard for []，返回整个数组。*不支持转义。

返回值说明

返回STRING类型的值。

说明

- 如果json为空或非法的json格式，返回NULL。
- 如果json合法，path也存在，则返回对应字符串。

示例代码

- 提取JSON对象src_json.json中的信息。命令示例如下。

  ```
  jsonString = {"store": {"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}], "bicycle":{"price":19.95,"color":"red"} }, "email":"amy@only_for_json_udf_test.net", "owner":"Tony" }
  ```

  提取owner字段信息，返回Tony。

  ```
  select get_json_object(jsonString, '$.owner');
  ```

  提取store.fruit字段第一个数组信息，返回{"weight":8,"type":"apple"}。

```
select get_json_object(jsonString, '$.store.fruit[0]');
```

提取不存在的字段信息，返回NULL。

```
select get_json_object(jsonString, '$.non_exist_key');
```

- 提取数组型JSON对象的信息。命令示例如下。

  返回22。

  ```
  select get_json_object('{"array":[["a",11],["b",22],["c",33]]}','$.array[1][1]');
  ```

  返回["h00","h11","h22"]。

  ```
  select get_json_object('{"a":"b","c":{"d":"e","f":"g","h":["h00","h11","h22"]},"i":"j"}','$.c.h[*]');
  ```

  返回["h00","h11","h22"]。

  ```
  select get_json_object('{"a":"b","c":{"d":"e","f":"g","h":["h00","h11","h22"]},"i":"j"}','$.c.h');
  ```

  返回h11。

  ```
  select get_json_object('{"a":"b","c":{"d":"e","f":"g","h":["h00","h11","h22"]},"i":"j"}','$.c.h[1]');
  ```

- 提取带有.的JSON对象中的信息。命令示例如下。

  创建一张表。

  ```
  create table json_table (id string, json string);
  ```

  向表中插入数据，Key带 "."

  ```
  insert into table json_table (id, json) values ("1", "{\"city1\":{\"region\":{\"rid\":6}}}");
  ```

  向表中插入数据，Key不带 "."

  ```
  insert into table json_table (id, json) values ("2", "{\"city1\":{\"region\":{\"rid\":7}}}");
  ```

  取rid的值，查询key为city1，返回6。由于包含.，只能用['']来解析。

  ```
  select get_json_object(json, "$['city1'].region['id']") from json_table where id =1;
  ```

  取rid的值，查询key为city1，返回7。查询方法有如下两种。

  ```
  select get_json_object(json, "$['city1'].region['id']") from json_table where id =2;
  select get_json_object(json, "$.city1.region['id']") from json_table where id =2;
  ```

- JSON输入为空或非法格式。命令示例如下。

  返回NULL。

  ```
  select get_json_object('','$.array[2]');
  ```

  返回NULL。

  ```
  select get_json_object('"array":["a",1],"b":["c",3]','$.array[1][1]');
  ```

- JSON字符串涉及转义。命令示例如下。

  返回"3"。

  ```
  select get_json_object('{"a":"\\"3\\"","b":"6"}', '$.a');
  ```

  返回'3'。

  ```
  select get_json_object('{"a":"\'3\'","b":"6"}', '$.a');
  ```

- 一个JSON对象中可以出现相同的Key，可以成功解析。

  返回1。

  ```
  select get_json_object('{"b":"1","b":"2"}', '$.b');
  ```

- 输出结果按照JSON字符串的原始排序方式输出。

  返回{"b":"3","a":"4"}。

  ```
  select get_json_object('{"b":"3","a":"4"}', '$');
  ```

1.29.2.9 instr

instr函数用于返回substr在str中最早出现的下标。

当参数中出现NULL时，返回NULL，当str中不存在substr时返回0，注意下标从1开始。

相似函数：**instr1**，instr1函数用于计算子串str2在字符串str1中的位置，instr1函数支持指定起始搜索位置和匹配次数。

命令格式

```
instr(string <str>, string <substr>)
```

参数说明

表 1-111 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待搜索的目标字符串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
substr	是	STRING	待匹配的子串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。

返回值说明

返回BIGINT类型的值。

说明

- 如果在str1中未找到str2，则返回0。
- 如果str2为空串，则总能匹配成功，例如select instr('abc','');会返回1。
- str1或str2值为NULL时，返回NULL。

示例代码

- 计算字符b在字符串abc中的位置。命令示例如下。
  返回2。
  ```
  select instr('abc', 'b');
  ```
- 任一输入参数为NULL。命令示例如下。
  返回NULL。
  ```
  select instr('abc', null)
  ```

1.29.2.10 instr1

instr1函数用于计算子串str2在字符串str1中的位置。

相似函数：**instr**，instr函数用于返回substr在str中最早出现的下标。但是instr不支持指定起始搜索位置和匹配次数。

命令格式

```
instr1(string <str1>, string <str2>[, bigint <start_position>[, bigint <nth_appearance>]])
```

参数说明

表 1-112 参数说明

参数	是否必选	参数类型	说明
str1	是	STRING	待搜索的目标字符串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
str2	是	STRING	待匹配的子串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
start_position	否	BIGINT	表示从str1的第几个字符开始搜索，默认起始位置是第一个字符位置1。 当start_position为负数时表示开始位置是从字符串的结尾往前倒数，最后一个字符是-1，依次往前倒数。
nth_appearance	否	BIGINT	表示str2在str1中第nth_appearance次匹配的位置。 如果nth_appearance为其他类型或小于等于0，则返回报错。

返回值说明

返回BIGINT类型。

说明

- 如果在str1中未找到str2，则返回0。
- 如果str2为空串，则总能匹配成功。
- str1、str2、start_position或nth_appearance值为NULL时，返回NULL。

示例代码

返回 10

```
select instr1('Tech on the net', 'h', 5, 1);
```

返回2。


```
select instr1('abc', 'b');
```

返回NULL。

```
select instr('abc', null);
```

1.29.2.11 initcap

initcap函数用于将文本字符串转换成首字母大写其余字母小写的形式。

命令格式

```
initcap(string A)
```

参数说明

表 1-113 参数说明

参数	是否必选	参数类型	说明
A	是	STRING	待转换的文本字符串。

返回值说明

返回一个STRING类型字符串，字符串中每个单词首字母大写，其余变为小写。

示例代码

返回Dli Sql

```
SELECT initcap("dLI sql");
```

1.29.2.12 keyvalue

keyvalue函数用于计算将字符串str按照split1进行切分，并按split2将每组变成Key-Value对，返回key所对应的Value。

命令格式

```
keyvalue(string <str>,[string <split1>,string <split2>,] string <key>)
```

参数说明

表 1-114 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待拆分的字符串。

参数	是否必选	参数类型	说明
split1、split2	否	STRING	用于作为分隔符的字符串，按照指定的两个分隔符拆分源字符串。如果表达式中没有指定这两项，默认split1为";"，split2为":"。当某个被split1拆分后的字符串中有多个split2时，返回结果未定义。
key	否	BIGINT	将字符串按照split1和split2拆分后，返回key值对应的Value。

返回值说明

返回STRING类型。

说明

- split1或split2值为NULL时，返回NULL。
- str或key值为NULL或没有匹配的key时，返回NULL。
- 如果有多个Key-Value匹配，返回第一个匹配上的key对应的Value。

示例代码

返回2。

```
select keyvalue('a:1;b:2', 'b');
```

返回2。

```
select keyvalue("\;abc:1\;def:2","\;",":","def");
```

1.29.2.13 length

length函数用于返回字符串的长度。

相似函数：**lengthb**，lengthb函数用于计算字符串str以字节为单位的长度，返回STRING类型的值。

命令格式

```
length(string <str>)
```


参数说明

表 1-115 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待搜索的目标字符串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。

返回值说明

返回BIGINT类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

- 计算字符串abc的长度。命令示例如下。
  返回3。
  ```
  select length('abc');
  ```
- 输入参数为NULL。命令示例如下。
  返回NULL。
  ```
  select length(null);
  ```

1.29.2.14 lengthb

lengthb函数用于计算字符串str以字节为单位的长度。

相似函数：**length**，length函数用于返回字符串的长度，返回BIGINT类型的值。

命令格式

```
lengthb(string <str>)
```

参数说明

表 1-116 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	输入的字符串。

返回值说明

返回STRING类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

返回5。

```
select lengthb('hello');
```

返回NULL。

```
select lengthb(null);
```

1.29.2.15 levenshtein

levenshtein函数用于返回两个字符串之间的Levenshtein距离，如levenshtein('kitten','sitting') =3。

说明

Levenshtein距离，是编辑距离的一种。指两个字串之间，由一个转成另一个所需的最少编辑操作次数。

命令格式

```
levenshtein(string A, string B)
```

参数说明

表 1-117 参数说明

参数	是否必选	参数类型	说明
A、B	是	STRING	计算Levenshtein距离需要输入的字符串。

返回值说明

返回INT类型的值。

示例代码

返回3

```
SELECT levenshtein('kitten','sitting');
```

1.29.2.16 locate

locate函数用于在str中查找substr的位置。您可以通过start_pos指定开始查找的位置，从1开始计数。

命令格式

```
locate(string <substr>, string <str>[, bigint <start_pos>])
```

参数说明

表 1-118 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待搜索的目标字符串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
substr	是	STRING	待匹配的子串。 如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算，其他类型会返回报错。
start_pos	否	BIGINT	指定查找的起始位置。

返回值说明

返回为BIGINT类型。

说明

- str中无法匹配到substr时，返回0。
- str或substr值为NULL时，返回NULL。
- start_pos值为NULL时，返回0。

示例代码

- 查找字符串ab在字符串abhiab中的位置。命令示例如下。
  返回1。
  ```
  select locate('ab', 'abhiab');
  ```
  返回5。
  ```
  select locate('ab', 'abhiab', 2);
  ```
  返回0。
  ```
  select locate('ab', 'abhiab', null);
  ```
- 查找字符串hi在字符串hanmeimei and lilei中的位置。命令示例如下。
  返回0。
  ```
  select locate('hi', 'hanmeimei and lilei');
  ```

1.29.2.17 lower/lcase

lower函数用于将文本字符串转换成字母全部小写的形式。

命令格式

```
lower(string A) / lcase(string A)
```

参数说明

表 1-119 参数说明

参数	是否必选	参数类型	说明
A	是	STRING	待转换的文本字符串。

返回值说明

返回为STRING类型的值。

说明

- 入参非 STRING、BIGINT、DOUBLE、DECIMAL 或 DATETIME 类型时，返回报错。
- 入参值为NULL时，返回NULL。

示例代码

将字符串中的大写字符转换为小写字符。命令示例如下。

返回 abc。

```
select lower('ABC');
```

输入参数为NULL。命令示例如下。

返回NULL。

```
select lower(null);
```

1.29.2.18 lpad

locate函数用于返回指定长度的字符串，给定字符串str1长度小于指定长度length时，由指定字符str2从左侧填补。

命令格式

```
lpad(string <str1>, int <length>, string <str2>)
```

参数说明

表 1-120 参数说明

参数	是否必选	参数类型	说明
str1	是	STRING	待向左补位的字符串。
length	是	STRING	向左补位位数。

参数	是否必选	参数类型	说明
str2	否	STRING	用于补位的字符串。

返回值说明

返回STRING类型的值。

说明

- 如果length小于str1的位数，则返回str1从左开始截取length位的字符串。
- 如果length为0，则返回空串。
- 如果没有输入参数或任一输入参数值为NULL，返回NULL。

示例代码

- 用字符串ZZ将字符串abcdefgh向左补足到10位。命令示例如下。
  返回ZZabcdefgh。
  ```
  select lpad('abcdefgh', 10, 'ZZ');
  ```
- 用字符串ZZ将字符串abcdefgh向左补足到5位。命令示例如下。
  返回abcde。
  ```
  select lpad('abcdefgh', 5, 'ZZ');
  ```
- length为0。命令示例如下。
  返回空串。
  ```
  select lpad('abcdefgh', 0, 'ZZ');
  ```
- 任一输入参数为NULL。命令示例如下。
  返回NULL。
  ```
  select lpad(null ,0, 'ZZ');
  ```

1.29.2.19 ltrim

ltrim函数用于从str的左端去除字符：

- 如果未指定trimChars，则默认去除空格字符。
- 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的左端去除尽可能长的所有字符都在集合trimChars中的子串。

相似函数：

- **rtrim**，rtrim函数用于从str的右端去除字符。
- **trim**，trim函数用于从str的左右两端去除字符。

命令格式

```
ltrim([<trimChars>,] string <str>)
```


参数说明

表 1-121 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待去除左端字符的字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
trimChars	是	STRING	待去除的字符。

返回值说明

返回为STRING类型。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str或trimChars值为NULL时，返回NULL。

示例代码

- 去除字符串" abc"的左边空格。命令示例如下。

  返回字符串abc 。

  ```
  select ltrim('    abc');
  ```

  等效于如下语句。

  ```
  select trim(leading from '    abc');
  ```

  leading代表去除字符串前面的空格
- 输入参数为NULL。命令示例如下。

  返回NULL。

  ```
  select ltrim(null);
  select ltrim('xy', null);
  select ltrim(null, 'xy');
  ```
- 去除字符串yxlucyxx左端所有字符都在集合xy中的子串。

  返回lucyxx，只要左端遇到x或者y就会被去掉。

  ```
  select ltrim( 'xy','yxlucyxx');
  ```

  等效于如下语句。

  ```
  select trim(leading 'xy' from 'yxlucyxx');
  ```

1.29.2.20 parse_url

parse_url函数用于返回给定URL的指定部分，partToExtract的有效值包括HOST，PATH， QUERY， REF， PROTOCOL， AUTHORITY，FILE和USERINFO。

例如：parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') 返回 'facebook.com'.。

当第二个参数为QUERY时，可以使用第三个参数提取特定参数的值，例如：parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') 返回'v1'。

命令格式

```
parse_url(string urlString, string partToExtract [, string keyToExtract])
```

参数说明

表 1-122 参数说明

参数	是否必选	参数类型	说明
urlString	是	STRING	URL链接。无效URL链接会返回报错。
partToExtract	是	STRING	取值包含：HOST、PATH、QUERY、REF、PROTOCOL、AUTHORITY、FILE和USERINFO，不区分大小写。
keyToExtract	否	STRING	当part取值为QUERY时，根据key值取出对应的Value值。

返回值说明

返回STRING类型。返回规则如下：

说明

- urlString、partToExtract或keyToExtract值为NULL时，返回NULL。
- partToExtract 取值不符合要求时，返回报错。

示例代码

返回example.com。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'HOST');
```

返回/over/there/index.dtb。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'PATH');
```

返回animal。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'QUERY', 'type');
```

返回nose。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'REF');
```

返回file。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'PROTOCOL');
```

返回 username@example.com:8042。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'AUTHORITY');
```

返回username。

```
select parse_url('file://username@example.com:666/over/there/index.dtb?
type=animal&name=narwhal#nose', 'USERINFO');
```

1.29.2.21 printf

printf函数用于将输入按特定格式打印输出。

命令格式

```
printf(String format, Obj... args)
```

参数说明

表 1-123 参数说明

参数	是否必选	参数类型	说明
format	是	STRING	用于定义输出格式
Obj	否	STRING	其他输入参数。

返回值说明

返回STRING类型的值。

将Obj中的参数填入format后打印输出。

示例代码;

返回字符串：姓名：user1，年龄：20，性别：女，籍贯：城市1。

```
SELECT printf('姓名：%s，年龄：%d，性别：%s，籍贯：%s', "user1", 20, "女", "城市1");
```

1.29.2.22 regexp_count

regexp_count函数用于计算source中从start_position位置开始，匹配指定pattern的子串数。

命令格式

```
regexp_count(string <source>, string <pattern>[, bigint <start_position>])
```

参数说明

表 1-124 参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待搜索的字符串，其他类型会返回报错。
pattern	是	STRING	STRING类型常量或正则表达式。待匹配的模型。pattern为空串或其他类型时返回报错。
start_position	否	BIGINT	BIGINT类型常量，必须大于0。其他类型或值小于等于0时返回报错。不指定时默认为1，表示从source的第一个字符开始匹配。

返回值说明

返回BIGINT类型的值。

说明

- 如果没有匹配成功，返回0。
- source、pattern值为NULL时，返回NULL。

示例代码

返回4。

```
select regexp_count('ab0a1a2b3c', '[0-9]');
```

返回3。

```
select regexp_count('ab0a1a2b3c', '[0-9]', 4);
```

返回 null。

```
select regexp_count('ab0a1a2b3c', null);
```

1.29.2.23 regexp_extract

REGEXP_EXTRACT函数用于将字符串source按照pattern的分组规则进行字符串匹配，返回第groupid个组匹配到的字符串内容。

命令格式

regexp_extract(string <source>, string <pattern>[, bigint <groupid>])

参数说明

表 1-125 参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待拆分的字符串。
pattern	是	STRING	STRING类型常量或正则表达式。待匹配的模型。
groupid	否	BIGINT	BIGINT类型常量，必须大于等于0。

返回值说明

返回STRING类型。

说明

- 如果pattern为空串或pattern中没有分组，返回报错。
- groupid非BIGINT类型或小于0时，返回报错。
- 不指定时默认为1，表示返回第一个组。
- 如果groupid等于0，则返回满足整个pattern的子串。
- source、pattern或groupid值为NULL时，返回NULL。

示例代码

将 basketball 按照 bas(.*?)(ball) 拆分。返回ket。

```
select regexp_extract('basketball', 'bas(.*?)(ball)');
```

返回 basketball 。

```
select regexp_extract('basketball', 'bas(.*?)(ball)',0);
```

返回99。在DLI上提交正则计算的SQL，需要使用两个"\"作为转义字符。

```
select regexp_extract('8d99d8', '8d(\\d+)d8');
```

返回【你好】。

```
select regexp_extract('【你好】hello', '([^\\x{00}-\\x{ff}]+)');
```

返回你好。

```
select regexp_extract('【你好】hello', '([\\x{4e00}-\\x{9fa5}]+)');
```

1.29.2.24 replace

replace函数用于用new字符串替换str字符串中与old字符串完全重合的部分并返回替换后的str。

如果没有重合的字符串，返回原str。

命令格式

```
replace(string <str>, string <old>, string <new>)
```

参数说明

表 1-126 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待替换的字符串。
old	是	STRING	待比较的字符串。
new	是	STRING	替换后的字符串。

返回值说明

返回STRING类型的值。

说明

如果任一输入参数值为NULL，返回NULL。

示例代码

返回AA123AA。

```
select replace('abc123abc','abc','AA');
```

返回NULL。

```
select replace('abc123abc',null,'AA');
```

1.29.2.25 regexp_replace

根据Spark版本不同，regexp_replace函数的功能略有差异：

- Spark2.4.5版本及以前版本：regexp_replace函数用于将source字符串中匹配pattern的子串替换成指定字符串replace_string后，返回结果字符串。
- Spark3.1.1版本：regexp_replace函数用于将source字符串中第occurrence次匹配pattern的子串，以及之后匹配pattern的子串，全都替换成指定字符串replace_string后，返回结果字符串。

相似函数：**regexp_replace1**，regexp_replace1函数用于将source字符串中第occurrence次匹配pattern的子串，替换成指定字符串replace_string后，返回结果字符串。但egexp_replace1函数仅适用于Spark2.4.5版本及以前版本。

即Spark2.4.5版适用的函数：regexp_replace1函数支持指定occurrence，但是regexp_replace函数不支持指定occurrence。

命令格式

- spark2.4.5及以前版本
  ```
  regexp_replace(string <source>, string <pattern>, string <replace_string>)
  ```
- spark3.1.1版本
  ```
  regexp_replace(string <source>, string <pattern>, string <replace_string>[, bigint <occurrence>])
  ```

参数说明

表 1-127 参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待替换的字符。
pattern	是	STRING	STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。pattern为空串时返回报错。
replace_string	是	STRING	将匹配pattern的字符串替换后的字符串。
occurrence	否	BIGINT	必须大于等于1，表示将第occurrence次匹配的字符串替换为replace_string，为1时表示替换所有匹配的子串。为其他类型或小于1时，返回报错。默认值为1。 **说明** 该字段仅Spark3.1.1版本的功能适用。

返回值说明

返回STRING类型的值。

说明

- 如果pattern为空串或pattern中没有分组，返回报错。
- 当引用不存在的组时，不进行替换。
- 如果replace_string值为NULL且pattern有匹配，返回NULL。
- 如果replace_string值为NULL但pattern不匹配，返回NULL。
- source、pattern或occurrence值为NULL时，返回NULL。

示例代码

- 适用于spark2.4.5及以前版本示例

  返回 num-num。

  ```
  SELECT regexp_replace('100-200', '(\\d+)', 'num');
  ```

- 适用于spark3.1.1版本示例。

  返回 2222。

  ```
  select regexp_replace('abcd', '[a-z]', '2');
  ```

  返回 2222。

  ```
  select regexp_replace('abcd', '[a-z]', '2', 1);
  ```

  返回 a222。

  ```
  select regexp_replace('abcd', '[a-z]', '2', 2);
  ```

  返回 ab22。

  ```
  select regexp_replace('abcd', '[a-z]', '2', 3);
  ```


返回 abc2。

```
select regexp_replace('abcd', '[a-z]', '2', 4);
```

1.29.2.26 regexp_replace1

regexp_replace1函数用于将source字符串中第occurrence次匹配pattern的子串，替换成指定字符串replace_string后，返回结果字符串。

说明

regexp_replace1函数只适用于Spark 2.4.5及之前的版本。

相似函数：**regexp_replace**，regexp_replace函数针对不同的Spark版本，功能略有差异，请参考**regexp_replace**查看详细的功能说明。

命令格式

```
regexp_replace1(string <source>, string <pattern>, string <replace_string>[, bigint <occurrence>])
```

参数说明

表 1-128 参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待替换的字符
pattern	是	STRING	STRING类型常量或正则表达式。待匹配的模型。更多正则表达式编写规范，请参见正则表达式规范。pattern为空串时返回报错。
replace_string	是	STRING	将匹配pattern的字符串替换后的字符串。
occurrence	否	BIGINT	必须大于等于1，表示将第occurrence次匹配的字符串替换为replace_string，为1时表示替换所有匹配的子串。为其他类型或小于1时，返回报错。默认值为1。

返回值说明

返回STRING类型的值。

说明

- 当引用不存在的组时，不进行替换。
- 如果replace_string值为NULL且pattern有匹配，返回NULL。
- 如果replace_string值为NULL但pattern不匹配，返回NULL。
- source、pattern或occurrence值为NULL时，返回NULL。

示例代码

返回 2222。

```
select regexp_replace1('abcd', '[a-z]', '2');
```

返回 2bcd。

```
select regexp_replace1('abcd', '[a-z]', '2', 1);
```

返回 a2cd。

```
select regexp_replace1('abcd', '[a-z]', '2', 2);
```

返回 ab2d。

```
select regexp_replace1('abcd', '[a-z]', '2', 3);
```

返回 abc2。

```
select regexp_replace1('abcd', '[a-z]', '2', 4);
```

1.29.2.27 regexp_instr

regexp_instr函数用于计算字符串source从start_position开始，与pattern第occurrence次匹配的子串的起始或结束位置。

命令格式

```
regexp_instr(string <source>, string <pattern>[,bigint <start_position>[, bigint <occurrence>[, bigint <return_option>]]])
```

参数说明

表 1-129 参数说明

参数	是否必选	参数类型	说明
source	是	STRING	源字符串。
pattern	是	STRING	STRING类型常量或正则表达式。待匹配的模型。pattern为空串时返回报错。
start_position	否	BIGINT	BIGINT类型常量。搜索的开始位置。不指定时默认值为1。
occurrence	否	BIGINT	BIGINT类型常量。指定匹配次数，不指定时默认值为1，表示搜索第一次出现的位置。
return_option	否	BIGINT	BIGINT类型常量。指定返回的位置。值为0或1，不指定时默认值为0，其他类型或不允许的值会返回报错。0表示返回匹配的开始位置，1表示返回匹配的结束位置。

返回值说明

返回BIGINT类型。return_option指定匹配的子串在source中的开始或结束位置。

说明

- 如果pattern为空串，返回报错。
- start_position或occurrence非BIGINT类型或小于等于0时，返回报错。
- source、pattern、start_position、occurrence或return_option值为NULL时，返回NULL

示例代码

返回6。

```
select regexp_instr('a1b2c3d4', '[0-9]', 3, 2);
```

返回NULL。

```
select regexp_instr('a1b2c3d4', null, 3, 2);
```

1.29.2.28 regexp_substr

regexp_substr函数用于计算从start_position位置开始，source中第occurrence次匹配指定pattern的子串。

命令格式

```
regexp_substr(string <source>, string <pattern>[, bigint <start_position>[, bigint <occurrence>]])
```

参数说明

表 1-130 参数说明

参数	是否必选	参数类型	说明
source	是	STRING	待搜索的字符串。
pattern	是	STRING	STRING类型常量或正则表达式。待匹配的模型。
start_position	否	BIGINT	起始位置，必须大于0。不指定时默认为1，表示从source的第一个字符开始匹配。
occurrence	否	BIGINT	BIGINT常量，必须大于0。不指定时默认为1，表示返回第一次匹配的子串。

返回值说明

返回STRING类型的值。

说明

- 如果pattern为空串，返回报错。
- 没有匹配时，返回NULL。
- start_position或occurrence非BIGINT类型或小于等于0时，返回报错。
- source、pattern、start_position、occurrence或return_option值为NULL时，返回NULL。

示例代码

返回a。

```
select regexp_substr('a1b2c3', '[a-z]');
```

返回b。

```
select regexp_substr('a1b2c3', '[a-z]', 2, 1);
```

返回c。

```
select regexp_substr('a1b2c3', '[a-z]', 2, 2);
```

返回NULL。

```
select regexp_substr('a1b2c3', null);
```

1.29.2.29 repeat

repeat函数用于返回将str重复n次后的字符串。

命令格式

```
repeat(string <str>, bigint <n>)
```

参数说明

表 1-131 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。
n	是	BIGINT	重复的数字n。

返回值说明

返回STRING类型。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- n为空时，返回报错。
- str或n值为NULL时，返回NULL。

示例代码

将字符‘123’重复2次，返回 123123。

```
SELECT repeat('123', 2);
```

1.29.2.30 reverse

reverse函数用于返回倒序字符串。

命令格式

```
reverse(string <str>)
```

参数说明

表 1-132 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	如果输入为BIGINT、DOUBLE、DECIMAL或DATETIME类型，则会隐式转换为STRING类型后参与运算。

返回值说明

返回STRING类型。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str值为NULL时，返回NULL。

示例代码

返回 LQS krapS。

```
SELECT reverse('Spark SQL');
```

返回[3,4,1,2]。

```
SELECT reverse(array(2, 1, 4, 3));
```

1.29.2.31 rpad

rpad函数用于将字符串str2将字符串str1向右补足到length位。

命令格式

```
rpad(string <str1>, int <length>, string <str2>)
```

参数说明

表 1-133 参数说明

参数	是否必选	参数类型	说明
str1	是	STRING	待向右补位的字符串。
length	是	INT	向右补位位数。
str2	是	STRING	用于补位的字符串。

返回值说明

返回STRING类型。

说明

- 如果length小于str1的位数，则返回str1从左开始截取length位的字符串。
- 如果length为0，则返回空串。
- 如果没有输入参数或任一输入参数值为NULL，返回NULL。

示例代码

返回 hi???。

```
SELECT rpad('hi', 5, '??');
```

返回 h。

```
SELECT rpad('hi', 1, '??');
```

1.29.2.32 rtrim

rtrim函数用于从str的右端去除字符：

- 如果未指定trimChars，则默认去除空格字符。
- 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的右端去除尽可能长的所有字符都在集合trimChars中的子串。

相似函数：

- **ltrim**，ltrim函数用于从str的左端去除字符。
- **trim**，trim函数用于从str的左右两端去除字符。

命令格式

```
rtrim([<trimChars>, ]string <str>)
```

或

```
rtrim(trailing [<trimChars>] from <str>)
```


参数说明

表 1-134 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待去除右端字符的字符串。如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
trimChars	是	STRING	待去除的字符。

返回值说明

返回为STRING类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str或trimChars值为NULL时，返回NULL。

示例代码

- 去除字符串 yxabcxx 的右边空格。命令示例如下。

  返回字符串 yxabcxx。

  ```
  select rtrim('yxabcxx     ');
  ```

  等效于如下语句。

  ```
  select trim(trailing from ' yxabcxx ');
  ```

- 去除字符串yxabcxx右端所有字符都在集合xy中的子串。

  返回yxabc，只要右端遇到x或者y就会被去掉。

  ```
  select rtrim('xy', 'yxabcxx');
  ```

  等效于如下语句。

  ```
  select trim(trailing 'xy' from 'yxabcxx');
  ```

- 输入参数为NULL。命令示例如下。

  返回NULL。

  ```
  select rtrim(null);
  select ltrim('yxabcxx', 'null');
  ```

1.29.2.33 soundex

soundex函数用于从str返回一个soundex字符串，如soundex('Miller')= M460。

命令格式

```
soundex(string <str>)
```

参数说明

表 1-135 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待转换的字符串。

返回值说明

返回STRING类型的值。

说明

str值为NULL时，返回NULL。

示例代码

返回M460

```
SELECT soundex('Miller');
```

1.29.2.34 space

space函数用于返回指定数量的空格。

命令格式

```
space(bigint <n>)
```

参数说明

表 1-136 参数说明

参数	是否必选	参数类型	说明
n	是	BIGINT	用于指定空格数量。

返回值说明

返回STRING类型。

说明

- n为空时，返回报错。
- n值为NULL时，返回NULL。

示例代码

返回6。

```
select length(space(6));
```

1.29.2.35 substr/substring

substr、substring函数用于返回字符串str从start_position开始，长度为length的子串。

命令格式

```
substr(string <str>, bigint <start_position>[, bigint <length>])
```

或

```
substring(string <str>, bigint <start_position>[, bigint <length>])
```

参数说明

表 1-137 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
start_position	是	BIGINT	表示起始位置。默认起始位置为1。 如果start_position取值为正，则从左边开始。如果start_position取值为负，则从右边开始。
length	否	BIGINT	表示子串的长度值。值必须大于0。

返回值说明

返回STRING类型的值。

说明

- str非STRING、BIGINT、DECIMAL、DOUBLE或DATETIME类型时，返回报错。
- length非BIGINT类型或值小于等于0时，返回报错。
- 当length被省略时，返回到str结尾的子串。
- str、start_position或length值为NULL时，返回NULL。

示例代码

返回 k SQL。

```
SELECT substr('Spark SQL', 5);
```

返回 SQL。

```
SELECT substr('Spark SQL', -3);
```

返回 k。

```
SELECT substr('Spark SQL', 5, 1);
```

1.29.2.36 substring_index

substring_index函数用于截取字符串str第count个分隔符之前的字符串。如果count为正，则从左边开始截取。如果count为负，则从右边开始截取。

命令格式

```
substring_index(string <str>, string <separator>, int <count>)
```

参数说明

表 1-138 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待截取的字符串。
separator	是	STRING	STRING类型的分隔符。
count	否	INT	指定分隔符位置。

返回值说明

返回STRING类型。

说明

如果任一输入参数值为NULL，返回NULL。

示例代码

返回 hello.world。

```
SELECT substring_index('hello.world.people', '.', 2);
```

返回world.people。

```
select substring_index('hello.world.people', '.', -2);
```

1.29.2.37 split_part

split_part函数用于依照分隔符separator拆分字符串str，返回从start部分到end部分的子串（闭区间）。

命令格式

```
split_part(string <str>, string <separator>, bigint <start>[, bigint <end>])
```

参数说明

表 1-139 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待拆分的字符串。
separator	是	STRING	STRING类型常量。拆分用的分隔符，可以是一个字符，也可以是一个字符串。
start	是	STRING	BIGINT类型常量，必须大于0。表示返回段的开始编号（从1开始）。
end	否	BIGINT	BIGINT类型常量，大于等于start。表示返回段的截止编号，可省略，缺省时表示和start取值相等，返回start指定的段。

返回值说明

返回STRING类型的值。

说明

- 如果start的值大于切分后实际的分段数，例如字符串拆分完有4个片段，start大于4，返回空串。
- 如果separator不存在于str中，且start指定为1，返回整个str。如果str为空串，则输出空串。
- 如果separator为空串，则返回原字符串str。
- 如果end大于片段个数，返回从start开始的子串。
- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- start或end非BIGINT类型常量时，返回报错。
- 除separator外，如果任一参数值为NULL，返回NULL。

示例代码

返回aa。

```
select split_part('aa,bb,cc,dd', ',', 1);
```

返回aa,bb。

```
select split_part('aa,bb,cc,dd', ',', 1, 2);
```

返回空串。

```
select split_part('aa,bb,cc,dd', ',', 10);
```

返回aa,bb,cc,dd。

```
select split_part('aa,bb,cc,dd', ':', 1);
```

返回空串。

```
select split_part('aa,bb,cc,dd', ':', 2);
```

返回aa,bb,cc,dd。


```
select split_part('aa,bb,cc,dd', '', 1);
```

返回bb,cc,dd。

```
select split_part('aa,bb,cc,dd', ',', 2, 6);
```

返回NULL。

```
select split_part('aa,bb,cc,dd', ',', null);
```

1.29.2.38 translate

translate函数用于将input字符串中的所出现的字符或者字符串from用字符或者字符串to替换。

例如：将abcde中的bcd替换成BCD。

```
translate("abcde", "bcd", "BCD")
```

命令格式

```
translate(string|char|varchar input, string|char|varchar from, string|char|varchar to)
```

参数说明

表 1-140 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待截取的字符串。
separator	是	STRING	STRING类型的分隔符。
count	否	INT	指定分隔符位置。

返回值说明

返回STRING类型的值。

说明

如果任一输入参数值为NULL，返回NULL。

示例代码

返回 A1B2C3。

```
SELECT translate('AaBbCc', 'abc', '123');
```

1.29.2.39 trim

trim函数用于从str的左右两端去除字符：

- 如果未指定trimChars，则默认去除空格字符。
- 如果指定了trimChars，则以trimChars中包含的字符作为一个集合，从str的左右两端去除尽可能长的所有字符都在集合trimChars中的子串。

相似函数：

- **ltrim**，ltrim函数用于从str的左端去除字符。
- **rtrim**，rtrim函数用于从str的右端去除字符。

命令格式

```
trim([<trimChars>,]string <str>)
```

或

```
trim([BOTH] [<trimChars>] from <str>)
```

参数说明

表 1-141 参数说明

参数	是否必选	参数类型	说明
str	是	STRING	待去除左右两端字符的字符串。 如果输入为BIGINT、DECIMAL、DOUBLE或DATETIME类型，则会隐式转换为STRING类型后参与运算。
trimChars	否	STRING	待去除的字符。

返回值说明

返回为STRING类型的值。

说明

- str非STRING、BIGINT、DOUBLE、DECIMAL或DATETIME类型时，返回报错。
- str或trimChars值为NULL时，返回NULL。

示例代码

- 去除字符串 yxabcxx 的左右空格。命令示例如下。
  返回字符串yxabcxx。
  ```
  select trim(' yxabcxx ');
  ```
  等效于如下语句。
  ```
  select trim(both from ' yxabcxx ');
  select trim(from ' yxabcxx ');
  ```
- 去除字符串yxabcxx左右两端所有字符都在集合xy中的子串。
  返回abc，只要左右两端遇到x或者y就会被去掉。
  ```
  select trim('xy', 'yxabcxx');
  ```
  等效于如下语句。
  ```
  select trim(both 'xy' from 'yxabcxx');
  ```
- 输入参数为NULL。命令示例如下。
  返回NULL。


```
select trim(null);
select trim(null, 'yxabcxx');
```

1.29.2.40 upper/ucase

upper函数用于从将文本字符串转换成字母全部大写的形式。

命令格式

```
upper(string A)
```

或

```
ucase(string A)
```

参数说明

表 1-142 参数说明

参数	是否必选	参数类型	说明
A	是	STRING	待转换的文本字符串。

返回值说明

返回STRING类型。

说明

- 入参非 STRING、BIGINT、DOUBLE、DECIMAL 或 DATETIME 类型时，返回报错。
- 入参值为NULL时，返回NULL。

示例代码

将字符串中的小写字符转换为大写字符。命令示例如下。

返回ABC。

```
select upper('abc');
```

输入参数为NULL。命令示例如下。

返回NULL。

```
select upper(null);
```

1.29.3 数学函数

1.29.3.1 数学函数概览

DLI所支持的数学函数如**数学函数**所示。

表 1-143 数学函数

命令格式	返回值	功能简介
abs(DOUBLE a)	DOUBLE或INT	取绝对值。
acos(DOUBLE a)	DOUBLE	返回给定角度a的反余弦值。
asin(DOUBLE a)	DOUBLE	返回给定角度a的反正弦值。
atan(DOUBLE a)	DOUBLE	返回给定角度a的反正切值。
bin(BIGINT a)	STRING	返回二进制格式。
bround(DOUBLE a)	DOUBLE	HALF_EVEN模式四舍五入，与传统四舍五入方式的区别在于，对数字5进行操作时，由前一位数字来决定，前一位数字为奇数，增加一位，前一位数字为偶数，舍弃一位。例如：bround(7.5)=8.0，bround(6.5)=6.0
bround(DOUBLE a, INT d)	DOUBLE	保留小数点后d位，d位之后数字以HALF_EVEN模式四舍五入。与传统四舍五入方式的区别在于，对数字5进行操作时，由前一位数字来决定，前一位数字为奇数，增加一位，前一位数字为偶数，舍弃一位。例如：bround(8.25, 1) = 8.2, bround(8.35, 1) = 8.4。
cbrt(DOUBLE a)	DOUBLE	返回a的立方根。
ceil(DOUBLE a)	DECIMAL	将参数向上舍入为最接近的整数。例如：ceil(21.2)，返回22。
conv(BIGINT num, INT from_base, INT to_base), conv(STRING num, INT from_base, INT to_base)	STRING	进制转换，将from_base进制下的num转化为to_base进制下面的数。例如：将5从十进制转换为四进制，conv(5,10,4)=11。
cos(DOUBLE a)	DOUBLE	返回给定角度a的余弦值。
cot1(DOUBLE a)	DOUBLE或DECIMAL类型	计算number的余切函数，输入为弧度值。
degrees(DOUBLE a)	DOUBLE	返回弧度所对应的角度。
e()	DOUBLE	返回e的值。
exp(DOUBLE a)	DOUBLE	返回e的a次方。
factorial(INT a)	BIGINT	返回a的阶乘。

命令格式	返回值	功能简介
floor(DOUBLE a)	BIGINT	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。
greatest(T v1, T v2, ...)	DOUBLE	返回列表中的最大值。
hex(BIGINT a) hex(STRING a)	STRING	将整数或字符转换为十六进制格式。
least(T v1, T v2, ...)	DOUBLE	返回列表中的最小值。
ln(DOUBLE a)	DOUBLE	返回给定数值的自然对数。
log(DOUBLE base, DOUBLE a)	DOUBLE	返回给定底数及指数返回自然对数。
log10(DOUBLE a)	DOUBLE	返回给定数值的以10为底自然对数。
log2(DOUBLE a)	DOUBLE	返回给定数值的以2为底自然对数。
median(colname)	DOUBLE或DECIMAL	计算中位数。
negative(INT a)	DECIMAL或INT	返回a的相反数，例如negative(2)，返回-2。
percentile(colname,DOUBLE p)	DOUBLE或ARRAY	计算精确百分位数，适用于小数据量。先对指定列升序排列，然后取精确的第p位百分数。p必须在0和1之间。
percentile_approx(colname,DOUBLE p)	DOUBLE或ARRAY	计算近似百分位数，适用于大数据量。先对指定列升序排列，然后取第p位百分数对应的值。
pi()	DOUBLE	返回pi的值。
pmod(INT a, INT b)	DECIMAL或INT	返回a除b的余数的绝对值。
positive(INT a)	DECIMAL、DOUBLE或INT	返回a的值，例如positive(2)，返回2。
pow(DOUBLE a, DOUBLE p), power(DOUBLE a, DOUBLE p)	DOUBLE	返回a的p次幂。
radians(DOUBLE a)	DOUBLE	返回角度所对应的弧度。

命令格式	返回值	功能简介
rand(INT seed)	DOUBLE	返回大于或等于0且小于1的平均分布随机数。如果指定种子seed，则会得到一个稳定的随机数序列。
round(DOUBLE a)	DOUBLE	四舍五入。
round(DOUBLE a, INT d)	DOUBLE	小数部分d位之后数字四舍五入，例如round(21.263,2)，返回21.26。
shiftleft(BIGINT a, INT b)	INT	有符号左边移，将a的二进制数按位左移b位。
shiftright(BIGINT a, INT b)	INT	有符号右移，将a的二进制数按位右移b位。
shiftrightunsigned(BIGINT a, INT b)	INT	无符号右移，将a的二进制数按位右移b位。
sign(DOUBLE a)	DOUBLE	返回a所对应的正负号，a为正返回1.0，a为负，返回-1.0，否则返回0.0。
sin(DOUBLE a)	DOUBLE	返回给定角度a的正弦值。
sqrt(DOUBLE a)	DOUBLE	返回数值的平方根。
tan(DOUBLE a)	DOUBLE	返回给定角度a的正切值。

1.29.3.2 abs

abs函数用于计算入参的绝对值。

命令格式

```
abs(DOUBLE a)
```

参数说明

表 1-144 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE或INT类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回NULL。

```
select abs(null);
```

返回1。

```
select abs(-1);
```

返回3.1415926。

```
select abs(-3.1415926);
```

1.29.3.3 acos

acos函数用于返回给定角度a的反余弦值。

命令格式

```
acos(DOUBLE a)
```

参数说明

表 1-145 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a取值范围为[-1,1]，a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型，值在0~π之间。

说明

- a的值不在[-1,1]范围内时，返回NaN。
- a为NULL，则返回NULL。

示例代码

返回3.141592653589793。

```
select acos(-1);
```

返回0。

```
select acos(1);
```

返回NULL。

```
select acos(null);
```

返回NAN。

```
select acos(10);
```

1.29.3.4 asin

asin函数用于返回给定角度a的反正弦值。

命令格式

```
asin(DOUBLE a)
```

参数说明

表 1-146 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a取值范围为[-1,1]，a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型，值在-π/2~π/2之间。

说明

- a的值不在[-1,1]范围内时，返回NaN。
- a为NULL，则返回NULL。

示例代码

返回1.5707963267948966。

```
select asin(1);
```

返回0.6435011087932844。

```
select asin(0.6);
```

返回NULL。

```
select asin(null);
```

返回NAN。

```
select asin(10);
```

1.29.3.5 atan

atan函数用于返回给定角度a的反正切值。

命令格式

```
atan(DOUBLE a)
```

参数说明

表 1-147 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型，值在-π/2~π/2之间。

说明

- a的值不在[-1,1]范围内时，返回NaN。
- a为NULL，则返回NULL。

示例代码

返回0.7853981633974483。

```
select atan(1);
```

返回0.5404195002705842。

```
select atan(0.6);
```

返回NULL。

```
select atan(null);
```

1.29.3.6 bin

bin函数用于返回a的二进制格式。

命令格式

```
bin(BIGINT a)
```

参数说明

表 1-148 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式为整数格式。 参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回STRING类型。

说明

a值为NULL时，返回NULL。

示例代码

返回1。

```
select bin(1);
```

返回NULL。

```
select bin(null);
```

返回1000。

```
select bin(8);
```

返回1000。

```
select bin(8.123456);
```

1.29.3.7 bround

bruond函数用于返回一个数值，该数值是按照指定d位小数进行四舍五入运算的结果。

命令格式

```
bround(DOUBLE a, INT d)
```

参数说明

表 1-149 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 代表需要被四舍五入的值。 该命令与传统四舍五入方式的区别在于，对数字5进行操作时，由前一位数字来决定，前一位数字为奇数，增加一位，前一位数字为偶数，舍弃一位。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。
d	否	DOUBLE、BIGINT、DECIMAL、STRING类型。	代表需要四舍五入到的位数。 参数d非INT类型时，会隐式转换为INT类型后参与运算。

返回值说明

返回DOUBLE类型。

说明

a或d值为NULL时，返回NULL。

示例代码

返回123.4。

```
select bround(123.45,1);
```

返回123.6。

```
select bround(123.55,1);
```

返回NULL。

```
select bround(null);
```

返回123.457。

```
select bround(123.456789,3.123456);
```

1.29.3.8 cbrt

cbrt函数用返回a的立方根。

命令格式

```
cbrt(DOUBLE a)
```

参数说明

表 1-150 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型。

说明

a为NULL，则返回NULL。

示例代码

返回3。

```
select cbrt(27);
```

返回3.3019272488946267。

```
select cbrt(36);
```

返回NULL。

```
select cbrt(null);
```

1.29.3.9 ceil

ceil函数用于对a进行向上舍入最接近的整数。

命令格式

```
ceil(DOUBLE a)
```

参数说明

表 1-151 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DECIMAL类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回2。

```
select ceil(1.3);
```

返回-1。

```
select ceil(-1.3);
```

返回NULL。

```
select ceil(null);
```

1.29.3.10 conv

conv函数用于进制转换，将from_base进制下的num转化为to_base进制下面的数。

命令格式

```
conv(BIGINT num, INT from_base, INT to_base)
```

参数说明

表 1-152 参数说明

参数	是否必选	参数类型	说明
num	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	需要进行转换进制的数。 参数num格式为浮点数格式、整数格式、字符串格式。
from_base	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	被转换的进制from_base。 参数from_base格式为浮点数格式、整数格式、字符串格式。
to_base	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	转化至的进制to_base。 参数to_base格式为浮点数格式、整数格式、字符串格式。

返回值说明

返回STRING类型。

说明

- num、from_base或to_base值为NULL时，返回NULL。
- 转换过程以64位精度工作，溢出时返回NULL。
- num如果输入的是小数，会转为整数值后进行进制转换，小数部分会被舍弃。

示例代码

-返回8。

```
select conv('1000', 2, 10);
```

返回B。

```
select conv('1011', 2, 16);
```

返回703710。

```
select conv('ABCDE', 16, 10);
```

返回27。

```
select conv(1000.123456, 3.123456, 10.123456);
```

返回18446744073709551589。

```
select conv(-1000.123456, 3.123456, 10.123456);
```

返回NULL。

```
select conv('1100', null, 10);
```

1.29.3.11 cos

cos函数用于计算a的余弦值，输入为弧度

命令格式

```
cos(DOUBLE a)
```

参数说明

表 1-153 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回-0.9999999999999986

```
select cos(3.1415926);
```

返回NULL。

```
select cos(null);
```

1.29.3.12 cot1

cot1函数用于计算a的余切值，输入为弧度。

命令格式

```
cot1(DOUBLE a)
```

参数说明

表 1-154 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE或DECIMAL类型。

说明

a为NULL，则返回NULL。

示例代码

返回1.0000000000000002。

```
select cot1(pi()/4);
```

返回NULL。

```
select cot1(null);
```

1.29.3.13 degrees

degress函数用于计算返回弧度所对应的角度。

命令格式

```
degrees(DOUBLE a)
```

参数说明

表 1-155 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回90.0。

```
select degrees(1.5707963267948966);
```

返回0。

```
select degrees(0);
```

返回NULL。

```
select degrees(null);
```

1.29.3.14 e

e函数用于计算返回e的值。

命令格式

```
e()
```

返回值说明

返回DOUBLE类型的值。

示例代码

返回2.718281828459045。

```
select e();
```

1.29.3.15 exp

abs函数用于计算返回e的a次方的值。

命令格式

```
exp(DOUBLE a)
```

参数说明

表 1-156 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回7.38905609893065。

```
select exp(2);
```

返回20.085536923187668。

```
select exp(3);
```

返回NULL。

```
select exp(null);
```

1.29.3.16 factorial

factorial函数用于返回a的阶乘。

命令格式

```
factorial(INT a)
```

参数说明

表 1-157 参数说明

参数	是否必选	参数类型	说明
a	是	BIGINT、INT、SMALLINT、TINYINT类型。	参数a的格式为整数格式。 参数a非INT类型时，会隐式转换为INT类型后参与运算。 字符串会转为对应的ASCII码。

返回值说明

返回BIGINT类型。

说明

- a值为0时，返回1。
- a值为NULL或[0,20]之外的值，返回NULL。

示例代码

返回720。

```
select factorial(6);
```

返回1。

```
select factorial(1);
```

返回120。

```
select factorial(5.123456);
```

返回NULL。

```
select factorial(null);
```

返回NULL。

```
select factorial(21);
```

1.29.3.17 floor

floor函数用于对a进行向下舍入最接近的整数。

命令格式

```
floor(DOUBLE a)
```


参数说明

表 1-158 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回BIGINT类型。

说明

a为NULL，则返回NULL。

示例代码

返回1。

```
select floor(1.2);
```

返回-2。

```
select floor(-1.2);
```

返回NULL。

```
select floor(null);
```

1.29.3.18 greatest

greatest函数用于返回列表中的最大值。

命令格式

```
greatest(T v1, T v2, ...)
```

参数说明

表 1-159 参数说明

参数	是否必选	参数类型	说明
v1	是	DOUBLE、BIGINT、DECIMAL类型。	参数v1的格式包括浮点数格式、整数格式。

参数	是否必选	参数类型	说明
v2	是	DOUBLE、BIGINT、DECIMAL类型。	参数v2的格式包括浮点数格式、整数格式。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回4.0。

```
select greatest(1,2.0,3,4.0);
```

返回NULL。

```
select greatest(null);
```

1.29.3.19 hex

hex函数用于将整数或字符转换为十六进制格式。

命令格式

```
hex(BIGINT a)
```

参数说明

表 1-160 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。 字符串会转为对应的ASCII码。

返回值说明

返回STRING类型。

说明

- a值为0时，返回0。
- a值为NULL时，返回NULL。

示例代码

返回0。

```
select hex(0);
```

返回61。

```
select hex('a');
```

返回10。

```
select hex(16);
```

返回31。

```
select hex('1');
```

返回3136。

```
select hex('16');
```

返回NULL。

```
select hex(null);
```

1.29.3.20 least

least函数用于返回列表中的最小值。

命令格式

```
least(T v1, T v2, ...)
```

参数说明

表 1-161 参数说明

参数	是否必选	参数类型	说明
v1	是	DOUBLE、BIGINT、DECIMAL类型。	参数v1的格式包括浮点数格式、整数格式。
v2	是	DOUBLE、BIGINT、DECIMAL类型。	参数v2的格式包括浮点数格式、整数格式。

返回值说明

返回DOUBLE类型的值。

说明

- v1、v2...为String类型时，返回报错。
- 所有参数都为NULL时，返回NULL。

示例代码

返回1.0。

```
select least(1,2.0,3,4.0);
```

返回NULL。

```
select least(null);
```

1.29.3.21 ln

ln函数用于返回给定数值的自然对数。

命令格式

```
ln(DOUBLE a)
```

参数说明

表 1-162 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

- a值为负数或0时，返回NULL。
- a值为NULL时，返回NULL。

示例代码

返回1.144729868791239。

```
select ln(3.1415926);
```

返回1。

```
select ln(2.718281828459045);
```

返回NULL。

```
select ln(null);
```

1.29.3.22 log

log函数根据给定底数及指数返回自然对数。

命令格式

```
log(DOUBLE base, DOUBLE a)
```

参数说明

表 1-163 参数说明

参数	是否必选	参数类型	说明
base	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数base的格式包括浮点数格式、整数格式、字符串格式。 参数base非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

- base或a为NULL时，返回NULL。
- base或a为负数或0时，返回NULL。
- 如果base为1（会引发一个除零行为），会返回NULL。

示例代码

返回2。

```
select log(2, 4);
```

返回NULL。

```
select log(2, null);
```

返回NULL。

```
select log(null, 4);
```

1.29.3.23 log10

log10函数用于返回给定数值的以10为底自然对数。

命令格式

```
log10(DOUBLE a)
```

参数说明

表 1-164 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a值为0、负数或NULL时，返回NULL。

示例代码

返回NULL。

```
select log10(null);
```

返回NULL。

```
select log10(0);
```

返回0.9542425094393249。

```
select log10(9);
```

返回1。

```
select log10(10);
```

1.29.3.24 log2

log2函数用于返回给定数值的以2为底自然对数。

命令格式

```
log2(DOUBLE a)
```

参数说明

表 1-165 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a值为0、负数或NULL时，返回NULL。

示例代码

返回NULL。

```
select log2(null);
```

返回NULL。

```
select log2(0);
```

返回3.1699250014423126。

```
select log2(9);
```

返回4。

```
select log2(16);
```

1.29.3.25 median

median函数用于计算入参的中位数。

命令格式

```
median(colname)
```


参数说明

表 1-166 参数说明

参数	是否必选	参数类型	说明
colname	是	DOUBLE、DECIMAL、STRING、BIGINT类型。	代表需要排序的列名。 列中元素为DOUBLE类型。 当列中元素非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE或DECIMAL类型。

说明

列名不存在时，返回报错。

示例代码

假设列int_test中的元素为1、2、3、4，类型为INT类型。

返回2.5。

```
select median(int_test) FROM int_test;
```

1.29.3.26 negative

negative函数用于返回a的相反数。

命令格式

```
negative(INT a)
```

参数说明

表 1-167 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回DECIMAL或INT类型。

说明

a为NULL，则返回NULL。

示例代码

返回-1。

```
SELECT negative(1);
```

返回3。

```
SELECT negative(-3);
```

1.29.3.27 percentlie

percentile函数用于计算精确百分位数，适用于小数据量。先对指定列升序排列，然后取第p位百分数的精确值。

命令格式

```
percentile(colname,DOUBLE p)
```

参数说明

表 1-168 参数说明

参数	是否必选	参数类型	说明
colname	是	STRING类型	代表需要排序的列名。 列中元素只能为整数类型。
p	是	DOUBLE类型	p的范围为0-1。参数p的格式包括浮点数格式。

返回值说明

返回DOUBLE或ARRAY类型。

说明

- 列名不存在时，返回报错。
- p为NULL或在[0,1]之外时，返回报错。

示例代码

假设列int_test中的元素为1、2、3、4，类型为INT类型。

返回3.0999999999999996。

```
select percentile(int_test,0.7) FROM int_test;
```

返回3.997。

```
select percentile(int_test,0.999) FROM int_test;
```

返回2.5。

```
select percentile(int_test,0.5) FROM int_test;
```

返回[1.3,1.9,2.5,2.8,3.7]。

```
select percentile (int_test,ARRAY(0.1,0.3,0.5,0.6,0.9)) FROM int_test;
```

1.29.3.28 percentlie_approx

percentile_approx函数用于计算近似百分位数，适用于大数据量。先对指定列升序排列，然后取第p位百分数最靠近的值。

命令格式

```
percentile_approx (colname,DOUBLE p)
```

参数说明

表 1-169 参数说明

参数	是否必选	参数类型	说明
colname	是	STRING类型	代表需要排序的列名。 列中元素为DOUBLE类型。当列中元素非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。
p	是	DOUBLE类型	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数p的范围为0-1。参数p的格式包括浮点数格式。

返回值说明

返回DOUBLE类型或ARRAY类型的值。

说明

- 列名不存在时，返回报错。
- p为NULL或在[0,1]之外时，返回报错。

示例代码

假设列int_test中的元素为1、2、3、4，类型为INT类型。

返回3。

```
select percentile_approx(int_test,0.7) FROM int_test;
```

返回3。

```
select percentile_approx(int_test,0.75) FROM int_test;
```

返回2。

```
select percentile_approx(int_test,0.5) FROM int_test;
```

返回[1,2,2,3,4]。

```
select percentile_approx (int_test,ARRAY(0.1,0.3,0.5,0.6,0.9)) FROM int_test;
```

1.29.3.29 pi

pi函数用于返回π的值。

命令格式

```
pi()
```

返回值说明

返回DOUBLE类型的值。

示例代码

返回3.141592653589793。

```
select pi();
```

1.29.3.30 pmod

pmod函数用于返回a除b的余数的绝对值。

命令格式

pmod(INT a, INT b)

参数说明

表 1-170 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数b的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回DECIMAL或INT类型。

说明

- a或b为NULL，则返回NULL。
- b为0时，返回NULL

示例代码

返回2。

```
select pmod(2,5);
```

返回3。

```
select pmod(-2,5)（解析：-2=5*(-1)...3）;
```

返回NULL。

```
select pmod(5,0);
```

返回1。

```
select pmod(5,2);
```

返回0.877。

```
select pmod(5.123,2.123);
```

1.29.3.31 positive

positive函数用于返回a的值。

命令格式

```
positive(INT a)
```

参数说明

表 1-171 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回 DECIMAL、DOUBLE或INT类型。

说明

a为NULL，则返回NULL。

示例代码

返回3。

```
SELECT positive(3);
```

返回-3。

```
SELECT positive(-3);
```

返回123。

```
SELECT positive('123');
```

1.29.3.32 pow

pow函数用于计算返回a的p次幂。

命令格式

```
pow(DOUBLE a, DOUBLE p),
power(DOUBLE a, DOUBLE p)
```

参数说明

表 1-172 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。
p	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数p的格式包括浮点数格式、整数格式、字符串格式。 参数p非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a、p为NULL，则返回NULL。

示例代码

返回16。

```
select pow(2, 4);
```

返回NULL。

```
select pow(2, null);
```

返回17.429460393524256。

```
select pow(2, 4.123456);
```

1.29.3.33 radians

radians函数用于返回角度所对应的弧度。

命令格式

```
radians(DOUBLE a)
```

参数说明

表 1-173 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回1.0471975511965976。

```
select radians(60);
```

返回0。

```
select radians(0);
```

返回NULL。

```
select radians(null);
```

1.29.3.34 rand

rand函数用于返回大于或等于0且小于1的平均分布随机数。

命令格式

```
rand(INT seed)
```

参数说明

表 1-174 参数说明

参数	是否必选	参数类型	说明
seed	否	INT类型。	参数seed的格式包括浮点数格式、整数格式、字符串格式。 如果指定种子seed，在相同运行环境下，将会得到一个稳定的随机数序列。

返回值说明

返回DOUBLE类型的值。

示例代码

返回0.3668915240363728。

```
select rand();
```

返回0.25738143505962285。

```
select rand(3);
```

1.29.3.35 round

round函数用于计算a的四舍五入到d位的值。

命令格式

```
round(DOUBLE a, INT d)
```

参数说明

表 1-175 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	代表需要被四舍五入的值。 参数a的格式包括浮点数格式、整数格式、字符串格式。
d	否	INT类型。	默认值：0。 代表需要四舍五入到的位数。 参数d非INT类型时，会隐式转换为INT类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

- d为负数时，返回报错。
- a或d值为NULL时，返回NULL。

示例代码

返回123.0。

```
select round(123.321);
```

返回123.4。

```
select round(123.396, 1);
```

返回NULL。

```
select round(null);
```

返回123.321。

```
select round(123.321, 4);
```

返回123.3。

```
select round(123.321,1.33333);
```

返回123.3。

```
select round(123.321,1.33333);
```

1.29.3.36 shiftleft

shiftleft函数用于有符号左移，将a的二进制数按位左移b位。

命令格式

shiftleft(BIGINT a, BIGINT b)

参数说明

表 1-176 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 当参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 当参数b非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回INT类型。

说明

a或b值为NULL时，返回NULL。

示例代码

返回8。

```
select shiftleft(1,3);
```

返回48。

```
select shiftleft(6,3);
```

返回48。

```
select shiftleft(6.123456,3.123456);
```

返回NULL。

```
select shiftleft(null,3);
```

1.29.3.37 shiftright

shiftright函数用于有符号右移，将a的二进制数按位右移b位。

命令格式

```
shiftright(BIGINT a, BIGINT b)
```

参数说明

表 1-177 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 当参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数b的格式包括浮点数格式、整数格式、字符串格式。 当参数b非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回INT类型。

说明

a或b值为NULL时，返回NULL。

示例代码

返回2。

```
select shiftright(16,3);
```

返回4。

```
select shiftright(36,3);
```

返回4。

```
select shiftright(36.123456,3.123456);
```

返回NULL。

```
select shiftright(null,3);
```

1.29.3.38 shiftrightunsigned

shiftrightunsigned函数用于无符号右移，将a的二进制数按位右移b位。

命令格式

```
shiftrightunsigned(BIGINT a, BIGINT b)
```

参数说明

表 1-178 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 当参数a非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。
b	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数b的格式包括浮点数格式、整数格式、字符串格式。 当参数b非BIGINT类型时，会隐式转换为BIGINT类型后参与运算。

返回值说明

返回INT类型。

说明

a或b值为NULL时，返回NULL。

示例代码

返回2。

```
select shiftrightunsigned(16,3);
```

返回536870910。

```
select shiftrightunsigned(-16,3);
```

返回2。

```
select shiftrightunsigned(16.123456,3.123456);
```

返回NULL。

```
select shiftrightunsigned(null,3);
```

1.29.3.39 sign

sign函数用于返回a所对应的正负号。

命令格式

```
sign(DOUBLE a)
```


参数说明

表 1-179 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。

返回值说明

返回DOUBLE类型。

说明

- a值为正数时，返回1。
- a值为负数时，返回-1。
- a值为0时，返回0。
- a值为NULL时，返回NULL。

示例代码

返回-1。

```
select sign(-3);
```

返回1。

```
select sign(3);
```

返回0。

```
select sign(0);
```

返回1。

```
select sign(3.1415926);
```

返回NULL。

```
select sign(null);
```

1.29.3.40 sin

sin函数用于计算a的正弦值，输入为弧度。

命令格式

```
sin(DOUBLE a)
```

参数说明

表 1-180 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回1。

```
select sin(pi()/2);
```

返回NULL。

```
select sin(null);
```

1.29.3.41 sqrt

sqrt函数用于返回数值的平方根。

命令格式

```
sqrt(DOUBLE a)
```

参数说明

表 1-181 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回2.8284271247461903。

```
select sqrt(8);
```

返回4。

```
select sqrt(16);
```

返回NULL。

```
select sqrt(null);
```

1.29.3.42 tan

tan函数用于计算a的正切值，输入为弧度。

命令格式

```
tan(DOUBLE a)
```

参数说明

表 1-182 参数说明

参数	是否必选	参数类型	说明
a	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	参数a的格式包括浮点数格式、整数格式、字符串格式。 参数a非DOUBLE类型时，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

返回0.9999999999999999。

```
select tan(pi()/4);
```

返回NULL。

```
select tan(null);
```

1.29.4 聚合函数

1.29.4.1 聚合函数概览

DLI所支持的聚合函数如**聚合函数表**所示。

表 1-183 聚合函数表

命令格式	返回值	功能简介
avg(col), avg(DISTINCT col)	DOUBLE	求平均值。
corr(col1, col2)	DOUBLE	返回两列数值的相关系数。
count([distinct|all] <colname>)	BIGINT	返回记录条数。
covar_pop(col1, col2)	DOUBLE	返回两列数值协方差。
covar_samp(col1, col2)	DOUBLE	返回两列数值样本协方差。
max(col)	DOUBLE	返回最大值。
min(col)	DOUBLE	返回最小值。
percentile(BIGINT col, p)	DOUBLE	返回数值区域的百分比数值点。0<=P<=1,否则返回NULL,不支持浮点型数值。
percentile_approx(DOUBLE col, p [, B])	DOUBLE	返回组内数字列近似的第p位百分数（包括浮点数），p值在[0,1]之间。参数B控制近似的精确度，B值越大，近似度越高，默认值为10000。当列中非重复值的数量小于B时，返回精确的百分数。
stddev_pop(col)	DOUBLE	返回指定列的偏差。
stddev_samp(col)	DOUBLE	返回指定列的样本偏差。
sum(col), sum(DISTINCT col)	DOUBLE	求和。
variance(col), var_pop(col)	DOUBLE	返回列的方差。
var_samp(col)	DOUBLE	返回指定列的样本方差。

1.29.4.2 avg

avg函数用于计算求平均值。

命令格式

```
avg(col), avg(DISTINCT col)
```

参数说明

表 1-184 参数说明

参数	是否必选	参数类型	说明
col	是	所有数据类型	列值支持所有数据类型，可以转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

如果col值为NULL时，该列不参与计算。

示例代码

- 计算所有仓库的平均商品数（items）。命令示例如下：

```
select avg(items) from warehouse;
```

返回结果如下：

```
_c0
100.0
```

- 与group by配合使用，计算每个仓库中所有商品的平均库存。命令示例如下：

```
select warehourseId, avg(items) from warehourse group by warehourseId;
```

返回结果如下：

```
warehouseId _c1
city1    155
city2    101
city3    194
```

1.29.4.3 corr

corr函数用于返回两列数值的相关系数。

命令格式

```
corr(col1, col2)
```

参数说明

表 1-185 参数说明

参数	是否必选	参数类型	说明
col1	是	DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL类型	数据类型为数值的列。其他类型返回NULL。
col2	是	DOUBLE、BIGINT、INT、SMALLINT、TINYINT、FLOAT、DECIMAL类型	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）和价格（price）的相关系数。命令示例如下：

```
select corr(items,price) from warehouse;
```

返回结果如下：

```
_c0
1.242355
```

- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）和价格（price）的相关系数。命令示例如下：

```
select warehourseId, corr(items,price) from warehourse group by warehourseId;
```

返回结果如下：

```
warehouseId _c1
city1   0.43124
city2   0.53344
city3   0.73425
```

1.29.4.4 count

count函数用于返回记录条数。

命令格式

```
count([distinct|all] <colname>)
```


参数说明

表 1-186 参数说明

参数	是否必选	说明
distinct或all	否	表示在计数时是否去除重复记录，默认为all，即计算全部记录。 如果指定distinct，则只计算唯一值数量。
colname	是	列值可以为任意类型。 colname可以为*，即count(*)，返回所有行数。

返回值说明

返回BIGINT类型。

说明

colname值为NULL时，该行不参与计算。

示例代码

- 计算所有仓库表中的记录数。命令示例如下：
```
select count(*) from warehouse;
```
返回结果如下：
```
_c0
10
```
- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，计算各仓库（warehouseId）的商品数。命令示例如下：
```
select warehouseId, count(*) from warehouse group by warehouseId;
```
返回结果如下：
```
warehouseId _c1
city1   6
city2   5
city3   6
```
示例3：通过distinct去重，计算仓库数量。命令示例如下：
```
select count(distinct warehouseId) from warehouse;
```
返回结果如下：
```
_c0
 3
```

1.29.4.5 covar_pop

covar_pop函数用于返回两列数值协方差。

命令格式

```
covar_pop(col1, col2)
```

参数说明

表 1-187 参数说明

参数	是否必选	说明
col1	是	数据类型为数值的列。其他类型返回NULL。
col2	是	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）和价格（price）的协方差。命令示例如下：
  ```
  select covar_pop(items，price) from warehouse;
  ```
  返回结果如下：
  ```
  _c0
  1.242355
  ```
- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）和价格（price）的协方差。命令示例如下：
  ```
  select warehourseId, covar_pop(items，price) from warehourse group by warehourseId;
  ```
  返回结果如下：
  ```
  warehouseId _c1
  city1    1.13124
  city2    1.13344
  city3    1.53425
  ```

1.29.4.6 covar_samp

covar_samp函数用于返回两列数值样本协方差。

命令格式

```
covar_samp(col1, col2)
```

参数说明

表 1-188 参数说明

参数	是否必选	说明
col1	是	数据类型为数值的列。其他类型返回NULL。
col2	是	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）和价格（price）的样本协方差。命令示例如下：
```
select covar_samp(items,price) from warehouse;
```
返回结果如下：
```
_c0
1.242355
```
- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）和价格（price）的样本协方差。命令示例如下：
```
select warehourseId, covar_samp(items,price) from warehourse group by warehourseId;
```
返回结果如下：
```
warehouseId _c1
city1    1.03124
city2    1.03344
city3    1.33425
```

1.29.4.7 max

max函数用于返回最大值。

命令格式

```
max(col)
```

参数说明

表 1-189 参数说明

参数	是否必选	参数类型	说明
col	是	除BOOLEAN外的任意类型。	列值可以为除BOOLEAN外的任意类型。

返回值说明

返回DOUBLE类型的值。

说明

返回值的类型与col类型相同。返回规则如下：

- col值为NULL时，该行不参与计算。
- col为BOOLEAN类型时，不允许参与运算。

示例代码

- 计算所有商品的最高库存（items）。命令示例如下：
```
select max(items) from warehouse;
```
返回结果如下：
```
_c0
900
```
- 与group by配合使用，求每个仓库的最高库存。命令示例如下：
```
select warehourseId, max(items) from warehouse group by warehourseId;
```

返回结果如下：

```
warehouseId _c1
city1    200
city2    300
city3    400
```

1.29.4.8 min

min函数用于返回最小值。

命令格式

```
min(col)
```

参数说明

表 1-190 参数说明

参数	是否必选	参数类型	说明
col	是	除BOOLEAN外的任意类型。	列值可以为除BOOLEAN外的任意类型。

返回值说明

返回DOUBLE类型的值。

说明

返回值的类型与col类型相同。返回规则如下：

- col值为NULL时，该行不参与计算。
- col为BOOLEAN类型时，不允许参与运算。

示例代码

- 计算所有商品的最低库存（items）。命令示例如下：

```
select min(items) from warehouse;
```

返回结果如下：

```
_c0
600
```

- 与group by配合使用，求每个仓库的最低库存。命令示例如下：

```
select warehourseId, min(items) from warehouse group by warehourseId;
```

返回结果如下：

```
warehouseId _c1
city1    15
city2    10
city3    19
```

1.29.4.9 percentile

percentlie函数用于返回数值区域的百分比数值点。

命令格式

```
percentile(BIGINT col, p)
```

参数说明

表 1-191 参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。其他类型返回NULL。
p	是	0<=P<=1,否则返回NULL。

返回值说明

返回DOUBLE类型的值。

说明

0<=P<=1,否则返回NULL。

示例代码

- 计算所有商品库存（items）的 0.5 百分位。命令示例如下：
  ```
  select percentile(items,0.5) from warehouse;
  ```
  返回结果如下：
  ```
  +------------+
  | _c0        |
  +------------+
  | 500.6      |
  +------------+
  ```
- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）的 0.5 百分位。命令示例如下：
  ```
  select warehourseId, percentile(items, 0.5) from warehourse group by warehourseId;
  ```
  返回结果如下：
  ```
  +------------+------------+
  | warehouseId| _c1        |
  +------------+------------+
  | city1    | 499.6      |
  | city2    | 354.8      |
  | city3    | 565.7      |
  +------------+------------+
  ```

1.29.4.10 percentile_approx

percentile_approx函数用于返回组内数字列近似的第p位百分数（包括浮点数）。

命令格式

```
percentile_approx(DOUBLE col, p [, B])
```

参数说明

表 1-192 参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。其他类型返回NULL。
p	是	0<=P<=1,否则返回NULL。
B	是	参数B控制近似的精确度，B值越大，近似度越高，默认值为10000。当列中非重复值的数量小于B时，返回精确的百分数。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）的 0.5 百分位，精确度100。命令示例如下：

```
select PERCENTILE_APPROX(items,0.5，100) from warehouse;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 521        |
+------------+
```

- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）的 0.5 百分位，精确度100。命令示例如下：

```
select warehourseId, PERCENTILE_APPROX(items, 0.5, 100) from warehourse group by warehourseId;
```

返回结果如下：

```
+------------+------------+
| warehouseId| _c1        |
+------------+------------+
| city1    | 499     |
| city2    | 354     |
| city3    | 565     |
+------------+------------+
```

1.29.4.11 stddev_pop

stddev_pop函数用于返回指定列的偏差。

命令格式

```
stddev_pop(col)
```


参数说明

表 1-193 参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）的偏差。命令示例如下：

```
select stddev_pop(items) from warehouse;
```

返回结果如下：

```
_c0
1.342355
```

- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）的偏差。命令示例如下：

```
select warehourseId, stddev_pop(items) from warehourse group by warehourseId;
```

返回结果如下：

```
warehouseId _c1
city1    1.23124
city2    1.23344
city3    1.43425
```

1.29.4.12 stddev_samp

stddev_samp函数用于返回指定列的样本偏差。

命令格式

```
stddev_samp(col)
```

参数说明

表 1-194 参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）的样本偏差。命令示例如下：

```
select stddev_samp(items) from warehouse;
```

返回结果如下：

```
+------------+
| _c0        |
+------------+
| 1.342355   |
+------------+
```

- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）的样本偏差。命令示例如下：

```
select warehourseId, stddev_samp(items) from warehourse group by warehourseId;
```

返回结果如下：

```
+------------+------------+
| warehouseId| _c1        |
+------------+------------+
| city1    | 1.23124   |
| city2    | 1.23344   |
| city3    | 1.43425   |
+------------+------------+
```

1.29.4.13 sum

sum函数用于计算求和。

命令格式

```
sum(col),
sum(DISTINCT col)
```

参数说明

表 1-195 参数说明

参数	是否必选	说明
col	是	列值支持所有数据类型，可以转换为DOUBLE类型后参与运算。 列值可以为DOUBLE、DECIMAL或BIGINT类型。 如果输入为STRING类型，会隐式转换为DOUBLE类型后参与运算。

返回值说明

返回DOUBLE类型的值。

说明

如果col值为NULL时，该行不参与计算。

示例代码

- 计算所有仓库的商品（items）总和。命令示例如下：

```
select sum(items) from warehouse;
```

返回结果如下：

```
_c0
55357
```

- 与group by配合使用，对所有商品按照仓库（warehouseId）进行分组，计算各仓库商品的总数（items）总和。命令示例如下：

```
select warehouseId, sum(items) from warehouse group by warehouseId;
```

返回结果如下：

```
warehouseId| _c1
city1    15500
city2    10175
city3    19400
```

1.29.4.14 variance/var_pop

variance/var_pop函数用于返回列的方差。

命令格式

```
variance(col),
var_pop(col)
```

参数说明

表 1-196 参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。 参数为其他类型的列返回NULL。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）的方差。命令示例如下：

```
select variance(items) from warehouse;
--等效于如下语句。
select var_pop(items) from warehouse;
```

返回结果如下：

```
_c0
203.42352
```

- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）的方差。命令示例如下：

```
select warehourseId, variance(items) from warehourse group by warehourseId;
--等效于如下语句。
select warehourseId, var_pop(items) from warehourse group by warehourseId;
```

返回结果如下：

```
warehouseId _c1
city1    19.23124
city2    17.23344
city3    12.43425
```

1.29.4.15 var_samp

var_samp函数用于返回指定列的样本方差。

命令格式

```
var_samp(col)
```

参数说明

表 1-197 参数说明

参数	是否必选	说明
col	是	数据类型为数值的列。 其他类型返回NULL。

返回值说明

返回DOUBLE类型的值。

示例代码

- 计算所有商品库存（items）的样本方差。命令示例如下：

  ```
  select var_samp(items) from warehouse;
  ```

  返回结果如下：

  ```
  _c0
  294.342355
  ```

- 与group by配合使用，对所有商品按照仓库（warehourseId）进行分组，并计算同组商品库存（items）的样本方差。命令示例如下：

  ```
  select warehourseId, var_samp(items) from warehourse group by warehourseId;
  ```

  返回结果如下：

  ```
  warehouseId _c1
  city1    18.23124
  city2    16.23344
  city3    11.43425
  ```

1.29.5 分析窗口函数

1.29.5.1 分析窗口函数概览

DLI所支持的分析窗口函数如**分析窗口函数介绍**所示。

表 1-198 分析窗口函数介绍

命令格式	返回值	功能简介
cume_dist()	DOUBLE	用于求累计分布，相当于求分区中大于等于或小于等于当前行的数据在分区中的占比。
first_value(col)	参数的数据类型	返回结果集中某列第一条数据的值。
last_value(col)	参数的数据类型	返回结果集中某列最后一条数据的值。
lag (col,n,DEFAULT)	参数的数据类型	用于统计窗口内往上第n行值。第一个参数为列名，第二个参数为往上第n行（可选，默认为1），第三个参数为默认值（当往上第n行为NULL时候，取默认值，如不指定，则为NULL）。
lead (col,n,DEFAULT)	参数的数据类型	用于统计窗口内往下第n行值。第一个参数为列名，第二个参数为往下第n行（可选，默认为1），第三个参数为默认值（当往下第n行为NULL时候，取默认值，如不指定，则为NULL）。
percent_rank()	DOUBLE	为窗口的ORDER BY子句所指定列中值的返回秩，但以介于0和1之间的小数形式表示，计算方法为 (RANK - 1)/(- 1)。
rank()	INT	计算一个值在一组值中的排位。如果出现并列的情况，RANK函数会在排名序列中留出空位。
row_number() over (order by col_1[,col_2 ...])	INT	为每一行指派一个唯一的编号。

1.29.5.2 cume_dist

cume_dist函数用于求累计分布，相当于求分区中大于等于或小于等于当前行的数据在分区中的占比。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
cume_dist() over([partition_clause] [orderby_clause])
```


参数说明

表 1-199 参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE salary (
dept STRING, -- 部门名称
userid string, -- 员工ID
sal INT -- 薪水
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
d1,user1,1000
d1,user2,2000
d1,user3,3000
d2,user4,4000
d2,user5,5000
```

- 统计小于等于当前薪水的人数占比

```
select dept, userid, sal,
      cume_dist() over(order by sal) as cume1
from salary;
--  结果：
d1 user1 1000 0.2
d1 user2 2000 0.4
d1 user3 3000 0.6
d2 user4 4000 0.8
d2 user5 5000 1.0
```

- 按部门分组统计小于等于当前薪水的人数的比例

```
select dept, userid, sal,
      cume_dist() over (partition by dept order by sal) as cume2
from salary;
-- 结果：
d1 user1 1000 0.3333333333333333
d1 user2 2000 0.6666666666666666
d1 user3 3000 1.0
d2 user4 4000 0.5
d2 user5 5000 1.0
```

- 按照sal降序排序后，结果就是统计 大于等于 当前薪水的人数的比例

```
select dept, userid, sal,
      cume_dist() over(order by sal desc) as cume3
```

```
from salary;
-- 结果：
d2 user5 5000 0.2
d2 user4 4000 0.4
d1 user3 3000 0.6
d1 user2 2000 0.8
d1 user1 1000 1.0
select dept, userid, sal,
      cume_dist() over(partition by dept order by sal desc) as cume4
from salary;
-- 结果：
d1 user3 3000 0.3333333333333333
d1 user2 2000 0.6666666666666666
d1 user1 1000 1.0
d2 user5 5000 0.5
d2 user4 4000 1.0
```

1.29.5.3 first_value

first_value函数用于取当前行所对应窗口的第一条数据的值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
first_value(<expr>[, <ignore_nulls>]) over ([partition_clause] [orderby_clause] [frame_clause])
```

参数说明

表 1-200 参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。
ignore_nulls	否	BOOLEAN类型，表示是否忽略NULL值。默认值为False。 当参数的值为True时，返回窗口中第一条非NULL的值。
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。
frame_clause	否	用于确定数据边界。

返回值说明

参数的数据类型。

示例代码

示例数据

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表logs，并添加数据，命令示例如下：

```
create table logs(
 cookieid string,
 createtime string,
 url string
)
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```

示例：将所有记录根据cookieid分组，并按createtime升序排列，返回每组中的第一行数据。命令示例如下

```
SELECT cookieid, createtime, url,
      FIRST_VALUE(url) OVER (PARTITION BY cookieid ORDER BY createtime) AS first
FROM logs;

返回结果如下：
cookieid createtime         url  first
cookie1 2015-04-10 10:00:00 url1 url1
cookie1 2015-04-10 10:00:02 url2 url1
cookie1 2015-04-10 10:03:04 url3 url1
cookie1 2015-04-10 10:10:00 url4 url1
cookie1 2015-04-10 10:50:01 url5 url1
cookie1 2015-04-10 10:50:05 url6 url1
cookie1 2015-04-10 11:00:00 url7 url1
cookie2 2015-04-10 10:00:00 url11 url11
cookie2 2015-04-10 10:00:02 url22 url11
cookie2 2015-04-10 10:03:04 url33 url11
cookie2 2015-04-10 10:10:00 url44 url11
cookie2 2015-04-10 10:50:01 url55 url11
cookie2 2015-04-10 10:50:05 url66 url11
cookie2 2015-04-10 11:00:00 url77 url11
```

1.29.5.4 last_value

last_value函数用于取当前行所对应窗口的最后一条数据的值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。

- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
last_value(<expr>[, <ignore_nulls>]) over ([partition_clause] [orderby_clause] [frame_clause])
```

参数说明

表 1-201 参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。
ignore_nulls	否	BOOLEAN类型，表示是否忽略NULL值。默认值为False。 当参数的值为True时，返回窗口中第一条非NULL的值。
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。
frame_clause	否	用于确定数据边界。

返回值说明

参数的数据类型。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表logs，并添加数据，命令示例如下：

```
create table logs(
 cookieid string,
 createtime string,
 url string
)
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```


示例：将所有记录根据cookieid分组，并按createtime升序排列，返回每组中的最后一行数据。命令示例如下

```
SELECT cookieid, createtime, url,
    LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS last
FROM logs;

-- 返回结果：
cookieid createtime         url last
cookie1 2015-04-10 10:00:00 url1 url1
cookie1 2015-04-10 10:00:02 url2 url2
cookie1 2015-04-10 10:03:04 url3 url3
cookie1 2015-04-10 10:10:00 url4 url4
cookie1 2015-04-10 10:50:01 url5 url5
cookie1 2015-04-10 10:50:05 url6 url6
cookie1 2015-04-10 11:00:00 url7 url7
cookie2 2015-04-10 10:00:00 url11 url11
cookie2 2015-04-10 10:00:02 url22 url22
cookie2 2015-04-10 10:03:04 url33 url33
cookie2 2015-04-10 10:10:00 url44 url44
cookie2 2015-04-10 10:50:01 url55 url55
cookie2 2015-04-10 10:50:05 url66 url66
cookie2 2015-04-10 11:00:00 url77 url77
```

说明

截止到当前行的最后一个值，其实就是它本身。

1.29.5.5 lag

lag函数用于用于统计窗口内往上第n行值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
lag(<expr>[, bigint <offset>[, <default>]]) over([partition_clause] orderby_clause)
```

参数说明

表 1-202 参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。
offset	否	偏移量，BIGINT类型常量，取值大于等于0。值为0时表示当前行，为1时表示前一行，以此类推。默认值为1。输入值为STRING类型、DOUBLE类型则隐式转换为BIGINT类型后进行运算。

参数	是否必选	说明
default	是	常量，默认值为NULL。 当offset指定的范围越界时的缺省值，需要与expr对应的数据类型相同。如果expr非常量，则基于当前行进行求值。
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

参数的数据类型。

示例代码

示例数据

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表logs，并添加数据，命令示例如下：

```
create table logs(
 cookieid string,
 createtime string,
 url string
)
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```

将所有记录根据cookieid分组，并按createtime升序排列，返回窗口内往上第2行的值。命令示例如下

示例1：

```
SELECT cookieid, createtime, url,
      LAG(createtime, 2) OVER (PARTITION BY cookieid ORDER BY createtime) AS last_2_time
FROM logs;
-- 返回结果：
cookieid createtime         url  last_2_time
cookie1 2015-04-10 10:00:00 url1 NULL
cookie1 2015-04-10 10:00:02 url2 NULL
cookie1 2015-04-10 10:03:04 url3 2015-04-10 10:00:00
cookie1 2015-04-10 10:10:00 url4 2015-04-10 10:00:02
cookie1 2015-04-10 10:50:01 url5 2015-04-10 10:03:04
```

```
cookie1 2015-04-10 10:50:05 url6 2015-04-10 10:10:00
cookie1 2015-04-10 11:00:00 url7 2015-04-10 10:50:01
cookie2 2015-04-10 10:00:00 url11 NULL
cookie2 2015-04-10 10:00:02 url22 NULL
cookie2 2015-04-10 10:03:04 url33 2015-04-10 10:00:00
cookie2 2015-04-10 10:10:00 url44 2015-04-10 10:00:02
cookie2 2015-04-10 10:50:01 url55 2015-04-10 10:03:04
cookie2 2015-04-10 10:50:05 url66 2015-04-10 10:10:00
cookie2 2015-04-10 11:00:00 url77 2015-04-10 10:50:01
```

说明

说明：因为没有设置默认值，当没有上两行时显示为NULL。

示例2：

```
SELECT cookieid, createtime, url,
       LAG(createtime,1,'1970-01-01 00:00:00') OVER (PARTITION BY cookieid ORDER BY createtime) AS
last_1_time
FROM cookie4;
-- 结果：
cookieid createtime          url last_1_time
cookie1 2015-04-10 10:00:00 url1 1970-01-01 00:00:00 (显示默认值)
cookie1 2015-04-10 10:00:02 url2 2015-04-10 10:00:00
cookie1 2015-04-10 10:03:04 url3 2015-04-10 10:00:02
cookie1 2015-04-10 10:10:00 url4 2015-04-10 10:03:04
cookie1 2015-04-10 10:50:01 url5 2015-04-10 10:10:00
cookie1 2015-04-10 10:50:05 url6 2015-04-10 10:50:01
cookie1 2015-04-10 11:00:00 url7 2015-04-10 10:50:05
cookie2 2015-04-10 10:00:00 url11 1970-01-01 00:00:00 (显示默认值)
cookie2 2015-04-10 10:00:02 url22 2015-04-10 10:00:00
cookie2 2015-04-10 10:03:04 url33 2015-04-10 10:00:02
cookie2 2015-04-10 10:10:00 url44 2015-04-10 10:03:04
cookie2 2015-04-10 10:50:01 url55 2015-04-10 10:10:00
cookie2 2015-04-10 10:50:05 url66 2015-04-10 10:50:01
cookie2 2015-04-10 11:00:00 url77 2015-04-10 10:50:05
```

1.29.5.6 lead

lead函数用于用于统计窗口内往下第n行值。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
lead(<expr>[, bigint <offset>[, <default>]]) over([partition_clause] orderby_clause)
```

参数说明

表 1-203 参数说明

参数	是否必选	说明
expr	是	待计算返回结果的表达式。

参数	是否必选	说明
offset	否	偏移量，BIGINT类型常量，取值大于等于0。值为0时表示当前行，为1时表示前一行，以此类推。默认值为1。输入值为STRING类型、DOUBLE类型则隐式转换为BIGINT类型后进行运算。
default	是	常量，默认值为NULL。 当offset指定的范围越界时的缺省值，需要与expr对应的数据类型相同。如果expr非常量，则基于当前行进行求值。
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

参数的数据类型。

示例代码

示例数据

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表logs，并添加数据，命令示例如下：

```
create table logs(
 cookieid string,
 createtime string,
 url string
)
STORED AS parquet;
```

添加数据如下：

```
cookie1 2015-04-10 10:00:02 url2
cookie1 2015-04-10 10:00:00 url1
cookie1 2015-04-10 10:03:04 url3
cookie1 2015-04-10 10:50:05 url6
cookie1 2015-04-10 11:00:00 url7
cookie1 2015-04-10 10:10:00 url4
cookie1 2015-04-10 10:50:01 url5
cookie2 2015-04-10 10:00:02 url22
cookie2 2015-04-10 10:00:00 url11
cookie2 2015-04-10 10:03:04 url33
cookie2 2015-04-10 10:50:05 url66
cookie2 2015-04-10 11:00:00 url77
cookie2 2015-04-10 10:10:00 url44
cookie2 2015-04-10 10:50:01 url55
```

将所有记录根据cookieid分组，并按createtime升序排列，返回窗口内往下第2行和第1行的值。命令示例如下

```
SELECT cookieid, createtime, url,
      LEAD(createtime, 2) OVER(PARTITION BY cookieid ORDER BY createtime) AS next_2_time,
      LEAD(createtime, 1, '1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER BY createtime) AS
next_1_time
FROM logs;
```

```
-- 返回结果：
cookieid createtime         url  next_2_time          next_1_time
cookie1 2015-04-10 10:00:00 url1 2015-04-10 10:03:04  2015-04-10 10:00:02
cookie1 2015-04-10 10:00:02 url2 2015-04-10 10:10:00  2015-04-10 10:03:04
cookie1 2015-04-10 10:03:04 url3 2015-04-10 10:50:01  2015-04-10 10:10:00
cookie1 2015-04-10 10:10:00 url4 2015-04-10 10:50:05  2015-04-10 10:50:01
cookie1 2015-04-10 10:50:01 url5 2015-04-10 11:00:00  2015-04-10 10:50:05
cookie1 2015-04-10 10:50:05 url6 NULL                 2015-04-10 11:00:00
cookie1 2015-04-10 11:00:00 url7 NULL                 1970-01-01 00:00:00
cookie2 2015-04-10 10:00:00 url11 2015-04-10 10:03:04 2015-04-10 10:00:02
cookie2 2015-04-10 10:00:02 url22 2015-04-10 10:10:00 2015-04-10 10:03:04
cookie2 2015-04-10 10:03:04 url33 2015-04-10 10:50:01 2015-04-10 10:10:00
cookie2 2015-04-10 10:10:00 url44 2015-04-10 10:50:05 2015-04-10 10:50:01
cookie2 2015-04-10 10:50:01 url55 2015-04-10 11:00:00 2015-04-10 10:50:05
cookie2 2015-04-10 10:50:05 url66 NULL                 2015-04-10 11:00:00
cookie2 2015-04-10 11:00:00 url77 NULL                 1970-01-01 00:00:00
```

1.29.5.7 percent_rank

percent_rank函数为窗口的ORDER BY子句所指定列中值的返回值，但以介于0和1之间的小数形式表示，计算方法为 (分组内当前行的RANK值-1)/(分组内总行数-1)。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
percent_rank() over([partition_clause] [orderby_clause])
```

参数说明

表 1-204 参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回DOUBLE类型的值。

示例代码

示例数据

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE salary (
dept STRING, -- 部门名称
userid string, -- 员工ID
sal INT -- 薪水
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
d1,user1,1000
d1,user2,2000
d1,user3,3000
d2,user4,4000
d2,user5,5000
```

示例：计算员工薪水在部门内的百分比排名。

```
select dept, userid, sal,
       percent_rank() over(partition by dept order by sal) as pr2
from salary;
-- 结果分析：
d1 user1 1000 0.0    -- (1-1)/(3-1)=0.0
d1 user2 2000 0.5    -- (2-1)/(3-1)=0.5
d1 user3 3000 1.0    -- (3-1)/(3-1)=1.0
d2 user4 4000 0.0    -- (1-1)/(2-1)=0.0
d2 user5 5000 1.0    -- (2-1)/(2-1)=1.0
```

1.29.5.8 rank

rank函数用于计算一个值在一组值中的排位。如果出现并列的情况，RANK函数会在排名序列中留出空位。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
rank() over ([partition_clause] [orderby_clause])
```

参数说明

表 1-205 参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回INT类型的值。

说明

a为NULL，则返回NULL。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。创建表logs，并添加数据，命令示例如下：

```
CREATE TABLE logs (
cookieid string,
createtime string,
pv INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
cookie1 2015-04-10 1
cookie1 2015-04-11 5
cookie1 2015-04-12 7
cookie1 2015-04-13 3
cookie1 2015-04-14 2
cookie1 2015-04-15 4
cookie1 2015-04-16 4
cookie2 2015-04-10 2
cookie2 2015-04-11 3
cookie2 2015-04-12 5
cookie2 2015-04-13 6
cookie2 2015-04-14 3
cookie2 2015-04-15 9
cookie2 2015-04-16 7
```

示例：将所有记录根据cookieid分组，并按pv降序排列，返回组内每行的序号。命令示例如下:

```
select cookieid, createtime, pv,
      rank() over(partition by cookieid order by pv desc) as rank
from logs
where cookieid = 'cookie1';
-- 结果：
cookie1 2015-04-12 7 1
cookie1 2015-04-11 5 2
cookie1 2015-04-16 4 3 (并列第三)
cookie1 2015-04-15 4 3
cookie1 2015-04-13 3 5 (跳过4，从5开始)
cookie1 2015-04-14 2 6
cookie1 2015-04-10 1 7
```

1.29.5.9 row_number

row_number函数用于计算行号。从1开始递增。

使用限制

窗口函数的使用限制如下：

- 窗口函数只能出现在select语句中。
- 窗口函数中不能嵌套使用窗口函数和聚合函数。
- 窗口函数不能和同级别的聚合函数一起使用。

命令格式

```
row_number() over([partition_clause] [orderby_clause])
```


参数说明

表 1-206 参数说明

参数	是否必选	说明
partition_clause	否	指定分区。分区列的值相同的行被视为在同一个窗口内。
orderby_clause	否	指定数据在一个窗口内如何排序。

返回值说明

返回DOUBLE类型的值。

说明

a为NULL，则返回NULL。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表logs，并添加数据，命令示例如下：

```
CREATE TABLE logs (
cookieid string,
createtime string,
pv INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
cookie1 2015-04-10 1
cookie1 2015-04-11 5
cookie1 2015-04-12 7
cookie1 2015-04-13 3
cookie1 2015-04-14 2
cookie1 2015-04-15 4
cookie1 2015-04-16 4
cookie2 2015-04-10 2
cookie2 2015-04-11 3
cookie2 2015-04-12 5
cookie2 2015-04-13 6
cookie2 2015-04-14 3
cookie2 2015-04-15 9
cookie2 2015-04-16 7
```

示例：将所有记录根据cookieid分组，并按pv降序排列，返回组内每行的序号。命令示例如下:

```
select cookieid, createtime, pv,
      row_number() over (partition by cookieid order by pv desc) as index
from logs;

-- 返回结果：
cookie1 2015-04-12 7 1
cookie1 2015-04-11 5 2
cookie1 2015-04-16 4 3
cookie1 2015-04-15 4 4
cookie1 2015-04-13 3 5
cookie1 2015-04-14 2 6
```

```
cookie1 2015-04-10 1 7
cookie2 2015-04-15 9 1
cookie2 2015-04-16 7 2
cookie2 2015-04-13 6 3
cookie2 2015-04-12 5 4
cookie2 2015-04-11 3 5
cookie2 2015-04-14 3 6
cookie2 2015-04-10 2 7
```

1.29.6 其他函数

1.29.6.1 函数概览

DLI提供了的decode1、javahash、max_pt等函数的说明如下。

表 1-207 其他新增函数说明

命令格式	返回值	功能简介
decode1(<expression>, <search>, <result>[, <search>, <result>]...[, <default>])	参数的数据类型	实现if-then-else分支选择的功能。
javahash(string a)	STRING	返回hash值。
max_pt(<table_full_name>)	STRING	返回分区表的一级分区中有数据的分区的最大值，按字母排序，且读取该分区下对应的数据。
ordinal(bigint <nth>, <var1>, <var2>[,...])	DOUBLE或DATETIME	将输入变量按从小到大排序后，返回nth指定位置的值。
trans_array (<num_keys>, <separator>, <key1>,<key2>, …,<col1>,<col2>,<col3>) as (<key1>,<key2>,...,<col1>, <col2>)	参数的数据类型	将一行数据转为多行的UDTF，将列中存储的以固定分隔符格式分隔的数组转为多行。
trunc_numeric(<number>[, bigint<decimal_places>])	DOUBLE或DECIMAL类型	将输入值number截取到指定小数点位置。
url_decode(string <input>[, string <encoding>])	STRING	将字符串从application/x-www-form-urlencoded MIME格式转为常规字符。

命令格式	返回值	功能简介
url_encode(string <input>[, string <encoding>])	STRING	将字符串编码为application/x-www-form-urlencoded MIME格式。

1.29.6.2 decode1

decode1函数实现if-then-else分支选择的功能。

命令格式

```
decode1(<expression>, <search>, <result>[, <search>, <result>]...[, <default>])
```

参数说明

表 1-208 参数说明

参数	是否必选	参数类型	说明
expression	是	所有数据类型。	要比较的表达式。
search	是	与expression一致。	与expression进行比较的搜索项。
result	是	所有数据类型。	search和expression的值匹配时的返回值。
default	否	与result一致。	如果所有的搜索项都不匹配，则返回default值，如果未指定，则返回NULL。

返回值说明

result 和 default 为返回值，支持返回所有的数据类型。

说明

- 如果匹配，返回result。
- 如果没有匹配，返回default。
- 如果没有指定default，返回NULL。
- 如果search选项有重复且匹配时，会返回第一个值。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE salary (
dept_id STRING, -- 部门
userid string, -- 员工ID
sal INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
d1,user1,1000
d1,user2,2000
d1,user3,3000
d2,user4,4000
d2,user5,5000
```

示例

返回每个部门的名称

当 dept_id 的值为d1时，返回DLI；值为d2时，返回MRS；其他场景返回Others。

```
select dept, decode1(dept, 'd1', 'DLI', 'd2', 'MRS', 'Others') as result from sale_detail;
```

返回结果：

```
d1 DLI
d2 MRS
d3 Others
d4 Others
d5 Others
```

1.29.6.3 javahash

javahash函数用于返回a的hash值。

命令格式

```
javahash(string a)
```

参数说明

表 1-209 参数说明

参数	是否必选	参数类型	说明
a	是	STRING类型。	需要返回hash值的数据。

返回值说明

返回STRING类型的值。

说明

返回hash值，如果a为null，返回报错。

示例代码

返回 48690

```
select javahash("123");
```

返回 123

```
select javahash(123);
```

1.29.6.4 max_pt

max_pt函数用于返回分区表的一级分区中有数据的分区的最大值，按字母排序，且读取该分区下对应的数据。

命令格式

```
max_pt(<table_full_name>)
```

参数说明

表 1-210 参数说明

参数	是否必选	参数类型	说明
table_full_name	是	STRING类型。	指定表名。必须对表有读权限。

返回值说明

返回STRING类型的值。

说明

- 返回最大的一级分区的值。
- 如果只是用alter table的方式新加了一个分区，但是此分区中并无任何数据，则此分区不会作为返回值。

示例代码

例如 table1 是分区表，该表对应的分区为20120801和20120802，且都有数据。则以下语句中max_pt返回值为‘20120802’。DLI SQL语句会读出pt=‘20120802’分区下的数据。

命令示例如下。

```
select * from tablel where pt = max_pt('dbname.table1');
```

等效于如下语句。

```
select * from table1 where pt = (select max(pt) from dbname.table1);
```

1.29.6.5 ordinal

ordinal函数用于将输入变量按从小到大排序后，返回nth指定位置的值。。

命令格式

```
ordinal(bigint <nth>, <var1>, <var2>[,...])
```

参数说明

表 1-211 参数说明

参数	是否必选	参数类型	说明
nth	是	BIGINT类型。	指定要返回的位置值。
var	是	BIGINT、DOUBLE、DATETIME或STRING类型。	待排序的值。

返回值说明

DOUBLE或DECIMAL类型。

说明

- 排在第nth位的值，当不存在隐式转换时返回值同输入参数数据类型。
- 当有类型转换时，DOUBLE、BIGINT、STRING之间的转换返回DOUBLE类型；STRING、DATETIME之间的转换返回DATETIME类型。不允许其他的隐式转换。
- NULL为最小值。

示例代码

返回2。

```
select ordinal(3, 1, 3, 2, 5, 2, 4, 9);
```

1.29.6.6 trans_array

trans_array函数用于将一行数据转为多行的UDTF，将列中存储的以固定分隔符格式分隔的数组转为多行。

使用限制

- 所有作为key的列必须位于在前面，而要转置的列必须放在后面。
- 在一个select中只能有一个UDTF，不可以再出现其他的列。
- 不可以与group by、cluster by、distribute by、sort by一起使用。

命令格式

```
trans_array (<num_keys>, <separator>, <key1>,<key2>,⋯,<col1>,<col2>,<col3>) as (<key1>,<key2>,...,<col1>,<col2>)
```

参数说明

表 1-212 参数说明

参数	是否必选	参数类型	说明
num_keys	是	BIGINT类型。	BIGINT类型常量，值必须>=0。在转为多行时作为转置key的列的个数。
separator	是	STRING类型。	STRING类型常量，用于将字符串拆分成多个元素的分隔符。为空时返回报错。
keys	是	STRING类型。	转置时作为key的列， 个数由num_keys指定。如果num_keys指定所有的列都作为key（即num_keys等于所有列的个数），则只返回一行。
cols	是	STRING类型。	要转为行的数组，keys之后的所有列视为要转置的数组，必须为STRING类型。

返回值说明

参数的数据类型。

说明

- 返回转置后的行，新的列名由as指定。
- 作为key的列类型保持不变，其余所有的列是STRING类型。
- 拆分成的行数以个数多的数组为准，不足的补NULL。

示例代码

为便于理解函数的使用方法，本文为您提供源数据，基于源数据提供函数相关示例。
创建表salary，并添加数据，命令示例如下：

```
CREATE EXTERNAL TABLE salary (
dept_id STRING, -- 部门
user_id string, -- 员工ID
sal INT -- 薪水
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
stored as textfile;
```

添加数据如下：

```
d1,user1/user4,1000/6000
d1,user2/user5,2000/7000
d1,user3/user6,3000
d2,user4/user7,4000
d2,user5/user8,5000/8000
```

执行SQL

```
select trans_array(1, "/", dept_id, user_id, sal) as (dept_id, user_id, sal) from salary;
```

返回结果如下：

```
d1,user1,1000
d1,user4,6000
d1,user2,2000
```

```
d1,user5,7000
d1,user3,3000
d1,user6,NULL
d2,user4,4000
d2,user7,NULL
d2,user5,5000
d2,user8,8000
```

1.29.6.7 trunc_numeric

trunc_numeric函数用于将输入值number截取到指定小数点位置。

命令格式

```
trunc_numeric(<number>[, bigint<decimal_places>])
```

参数说明

表 1-213 参数说明

参数	是否必选	参数类型	说明
number	是	DOUBLE、BIGINT、DECIMAL、STRING类型。	需要截取的数据。
decimal_places	否	BIGINT类型。	默认为0，截取位置的小数点位。

返回值说明

返回DOUBLE或DECIMAL类型。

说明

返回规则如下：

- number为DOUBLE、DECIMAL类型时会返回相应的类型。
- number为STRING、BIGINT类型时，返回DOUBLE类型。
- decimal_places非BIGINT类型时，返回报错。
- number值为NULL时，返回NULL。

示例代码

返回 3.141。

```
select trunc_numeric(3.1415926, 3);
```

返回 3。

```
select trunc_numeric(3.1415926);
```

报错。

```
select trunc_numeric(3.1415926, 3.1);
```

1.29.6.8 url_decode

url_decode函数用于将字符串从application/x-www-form-urlencoded MIME格式转为常规字符。

命令格式

```
url_decode(string <input>[, string <encoding>])
```

参数说明

表 1-214 参数说明

参数	是否必选	参数类型	说明
input	是	STRING类型。	要输入的字符串。
endcoding	否	STRING类型。	指定编码格式，支持GBK或UTF-8等标准编码格式，不输入默认为UTF-8。

返回值说明

返回STRING类型的值。

说明

STRING类型UTF-8编码的字符串。

示例代码

返回 Example for URL_DECODE:// dsf(fasfs)。

```
select url_decode('Example+for+url_decode+%3A%2F%2F+dsf%28fasfs%29', 'GBK');
```

1.29.6.9 url_encode

url_endcode函数用于将字符串编码为application/x-www-form-urlencoded MIME格式。

命令格式

url_encode(string <input>[, string <encoding>])

参数说明

表 1-215 参数说明

参数	是否必选	参数类型	说明
input	是	STRING类型。	要输入的字符串。
endcoding	否	STRING类型。	指定编码格式，支持GBK或UTF-8等标准编码格式，不输入默认为UTF-8。

返回值说明

返回STRING类型的值。

说明

input或encoding值为NULL时，返回NULL。

示例代码

返回Example+for+url_encode+%3A%2F%2F+dsf%28fasfs%29。

```
select url_encode('Example for url_encode:// dsf(fasfs)', 'GBK');
```

1.30 SELECT 基本语句

功能描述

基本的查询语句，返回查询结果。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_name_list]
  [ORDER BY col_name_list][ASC | DESC]
  [CLUSTER BY col_name_list | DISTRIBUTE BY col_name_list]
  [SORT BY col_name_list]]
  [LIMIT number];
```

关键字

表 1-216 SELECT 关键字说明

参数	描述
ALL	ALL关键字用于返回数据库所有匹配的行，包括重复的行。ALL关键字的后面只能跟*，否则执行语句会出错。 ALL是SQL语句的默认行为，通常不会被明确写出，如果不指定ALL或DISTINCT，查询结果将包含所有的行，即使是重复的行数据也将被返回。
DISTINCT	在SELECT语句中使用DISTINCT关键字时，系统会在查询结果中去除重复的数据，确保结果的唯一性。
WHERE	指定查询的过滤条件，支持算术运算符、关系运算符和逻辑运算符。
where_condition	过滤条件。
GROUP BY	指定分组的字段，支持单字段及多字段分组。
col_name_list	字段列表。
ORDER BY	对查询结果进行排序。
ASC/DESC	ASC为升序，DESC为降序，默认为ASC。
CLUSTER BY	为分桶且排序，按照分桶字段先进行分桶，再在每个桶中依据该字段进行排序，即当DISTRIBUTE BY的字段与SORT BY的字段相同且排序为降序时，两者的作用与CLUSTER BY等效。
DISTRIBUTE BY	指定分桶字段，不进行排序。
SORT BY	将会在桶内进行排序。
LIMIT	对查询结果进行限制，number参数仅支持INT类型。

注意事项

- 所查询的表必须是已经存在的表，否则提示查询错误。
- 在DLI管理控制台提交SQL语句读取binary类型的数据进行展示时，会对binary数据进行Base64转换。

示例

将表student中，name为Mike的数据记录查询出来，并根据字段score升序排序。

```
SELECT * FROM student
  WHERE name = 'Mike'
  ORDER BY score;
```

1.31 过滤 SELECT

1.31.1 WHERE 过滤子句

功能描述

利用WHERE子句过滤查询结果。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference
  WHERE where_condition;
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。
- WHERE：条件过滤关键字，将不满足条件的记录过滤掉，返回满足要求的记录。

注意事项

所查询的表必须是已经存在的，否则会出错。

示例

将表student中，score在（90，95）之间的记录筛选出来。

```
SELECT * FROM student
  WHERE score > 90 AND score < 95;
```

1.31.2 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] attr_expr_list FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_name_list]
  HAVING having_condition;
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。
- 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

- 所查询的表必须是已经存在的，否则会出错。
- 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。

```
SELECT name, max(score) FROM student
  GROUP BY name
  HAVING max(score) >95;
```

1.32 排序 SELECT

1.32.1 ORDER BY

功能描述

按字段实现查询结果的全局排序。

语法格式

```
SELECT attr_expr_list FROM table_reference
  ORDER BY col_name
  [ASC | DESC] [,col_name [ASC | DESC],...];
```

关键字

- ASC/DESC：ASC为升序，DESC为降序，默认为ASC。
- ORDER BY：对全局进行单列或多列排序。与GROUP BY一起使用时，ORDER BY后面可以跟聚合函数。

注意事项

所排序的表必须是已经存在的，否则会出错。

示例

根据字段score对表student进行升序排序，并返回排序后的结果。

```
SELECT * FROM student
  ORDER BY score;
```

1.32.2 SORT BY

功能描述

按字段实现表的局部排序。

语法格式

```
SELECT attr_expr_list FROM table_reference
  SORT BY col_name
  [ASC | DESC] [,col_name [ASC | DESC],...];
```

关键字

- ASC/DESC：ASC为升序，DESC为降序，默认为ASC。
- SORT BY：一般与GROUP BY一起使用，为PARTITION进行单列或多列的局部排序。

注意事项

所排序的表必须是已经存在的，否则会出错。

示例

根据字段score对表student在Reducer中进行升序排序。

```
SELECT * FROM student
  SORT BY score;
```

1.32.3 CLUSTER BY

功能描述

按字段实现表的分桶及桶内排序。

语法格式

```
SELECT attr_expr_list FROM table_reference
  CLUSTER BY col_name [,col_name ,...];
```

关键字

CLUSTER BY：根据指定的字段进行分桶，支持单字段及多字段，并在桶内进行排序。

注意事项

所排序的表必须是已经存在的，否则会出错。

示例

根据字段score对表student进行分桶并进行桶内局部降序排序。

```
SELECT * FROM student
  CLUSTER BY score;
```

1.32.4 DISTRIBUTE BY

功能描述

按字段实现表的分桶。

语法格式

```
SELECT attr_expr_list FROM table_reference
  DISTRIBUTE BY col_name [,col_name ,...];
```

关键字

DISTRIBUTE BY：根据指定的字段进行分桶，支持单字段及多字段，不会在桶内进行排序。与SORT BY配合使用即为分桶后的排序。

注意事项

所排序的表必须是已经存在的，否则会出错。

举例

根据字段score对表student进行分桶。

```
SELECT * FROM student
  DISTRIBUTE BY score;
```

1.33 分组 SELECT

1.33.1 按列 GROUP BY

功能描述

按列对表进行分组操作。

语法格式

```
SELECT attr_expr_list FROM table_reference
  GROUP BY col_name_list;
```

关键字

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，col_name_list中包含的字段必须出现在attr_expr_list的字段内，attr_expr_list中可以使用多个聚合函数，比如count()，sum()，聚合函数中可以包含其他字段。
- 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，同样，GROUP BY中出现的字段必须在attr_expr_list的字段内，attr_expr_list也可以使用聚合函数。

注意事项

所要分组的表必须是已经存在的表，否则会出错。

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。

```
SELECT score, count(name) FROM student
  GROUP BY score,name;
```

1.33.2 用表达式 GROUP BY

功能描述

按表达式对表进行分组操作。

语法格式

```
SELECT attr_expr_list FROM table_reference
  GROUP BY groupby_expression [, groupby_expression, ...];
```

关键字

groupby_expression：可以是单字段，多字段，也可以是聚合函数，字符串函数等。

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 同单列分组，GROUP BY中出现的字段必须包含在attr_expr_list的字段中，表达式支持内置函数，自定义函数等。

示例

先利用substr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。

```
SELECT substr(name,6),count(name) FROM student
  GROUP BY substr(name,6);
```

1.33.3 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 attr_expr_list FROM table_reference
  GROUP BY groupby_expression[, groupby_expression… ]
  HAVING having_expression;
```

关键字

groupby_expression：可以是单字段，多字段，也可以是聚合函数，字符串函数等。

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 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子句中可支持算术运算，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。

```
SELECT num, max(price*amount) FROM transactions
  WHERE time > '2016-06-01'
  GROUP BY num
  HAVING max(price*amount)>5000;
```

1.33.4 ROLLUP

功能描述

ROLLUP生成聚合行、超聚合行和总计行。可以实现从右到左递减多级的统计，显示统计某一层次结构的聚合。

语法格式

```
SELECT attr_expr_list FROM table_reference
  GROUP BY col_name_list
  WITH ROLLUP;
```

关键字

ROLLUP：为GROUP BY的扩展，例如：***SELECT a, b, c, SUM(expression) FROM table GROUP BY a, b, c WITH ROLLUP;***将转换成以下四条查询：

- (a, b, c)组合小计
  ```
  SELECT a, b, c, sum(expression) FROM table
    GROUP BY a, b, c;
  ```
- (a, b)组合小计
  ```
  SELECT a, b, NULL, sum(expression) FROM table
    GROUP BY a, b;
  ```
- (a)组合小计
  ```
  SELECT a, NULL, NULL, sum(expression) FROM table
    GROUP BY a;
  ```
- 总计
  ```
  SELECT NULL, NULL, NULL, sum(expression) FROM table;
  ```

注意事项

所要分组的表必须是已经存在的表，否则会出错。

示例

根据group_id与job两个字段生成聚合行、超聚合行和总计行，返回每种聚合情况下的salary总和。

```
SELECT group_id, job, SUM(salary) FROM group_test
  GROUP BY group_id, job
  WITH ROLLUP;
```

1.33.5 GROUPING SETS

功能描述

GROUPING SETS生成交叉表格行，可以实现GROUP BY字段的交叉统计。

语法格式

```
SELECT attr_expr_list FROM table_reference
  GROUP BY col_name_list
  GROUPING SETS(col_name_list);
```

关键字

GROUPING SETS：为对GROUP BY的扩展，例如

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS((a,b));***

  将转换为以下一条查询：

  ```
  SELECT a, b, sum(expression) FROM table
    GROUP BY a, b;
  ```

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS(a,b);***

  将转换为以下两条查询：

  ```
  SELECT a, NULL, sum(expression) FROM table GROUP BY a;
  UNION
  SELECT NULL, b, sum(expression) FROM table GROUP BY b;
  ```

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS((a,b), a);***

  将转换为以下两条查询：

  ```
  SELECT a, b, sum(expression) FROM table GROUP BY a, b;
  UNION
  SELECT a, NULL, sum(expression) FROM table GROUP BY a;
  ```

- ***SELECT a, b, sum(expression) FROM table GROUP BY a, b GROUPING SETS((a,b), a, b, ());***

  将转换为以下四条查询：

  ```
  SELECT a, b, sum(expression) FROM table GROUP BY a, b;
  UNION
  SELECT a, NULL, sum(expression) FROM table GROUP BY a, NULL;
  UNION
  SELECT NULL, b, sum(expression) FROM table GROUP BY NULL, b;
  UNION
  SELECT NULL, NULL, sum(expression) FROM table;
  ```

注意事项

- 所要分组的表必须是已经存在的表，否则会出错。
- 不同于ROLLUP，GROUPING SETS目前仅支持一种格式。

示例

根据group_id与job两个字段生成交叉表格行，返回每种聚合情况下的salary总和。

```
SELECT group_id, job, SUM(salary) FROM group_test
  GROUP BY group_id, job
  GROUPING SETS (group_id, job);
```


1.34 连接操作 SELECT

1.34.1 内连接

功能描述

仅将两个表中满足连接条件的行组合起来作为结果集。

语法格式

```
SELECT attr_expr_list FROM table_reference
  {JOIN | INNER JOIN} table_reference ON join_condition;
```

关键字

JOIN/INNER JOIN：只显示参与连接的表中满足JOIN条件的记录。

注意事项

- 所要进行JOIN连接的表必须是已经存在的表，否则会出错。
- 在一次查询中可以连接两个以上的表。

示例

通过将student_info与course_info两张表中的课程编号匹配建立JOIN连接，来查看学生姓名及所选课程名称。

```
SELECT student_info.name, course_info.courseName FROM student_info
  JOIN course_info ON (student_info.courseId = course_info.courseId);
```

1.34.2 左外连接

功能描述

根据左表的记录去匹配右表，返回所有左表记录，没有匹配值的记录的返回NULL。

语法格式

```
SELECT attr_expr_list FROM table_reference
  LEFT OUTER JOIN table_reference ON join_condition;
```

关键字

LEFT OUTER JOIN：返回左表的所有记录，没有匹配值的记录将返回NULL。

注意事项

所要进行JOIN连接的表必须是已经存在的表，否则会出错。

示例

左外连接时利用student_info表中的courseId与course_info中的courseId进行匹配，返回已经选课的学生姓名及所选的课程名称，没有匹配值的右表记录将返回NULL。

```
SELECT student_info.name, course_info.courseName FROM student_info
  LEFT OUTER JOIN course_info ON (student_info.courseId = course_info.courseId);
```

1.34.3 右外连接

功能描述

根据右表的记录去匹配左表，返回所有右表记录，没有匹配值的记录返回NULL。

语法格式

```
SELECT attr_expr_list FROM table_reference
  RIGHT OUTER JOIN table_reference ON join_condition;
```

关键字

RIGHT OUTER JOIN：返回右表的所有记录，没有匹配值的记录将返回NULL。

注意事项

所要进行JOIN连接的表必须是已经存在的表，否则会出错。

示例

右外连接和左外连接相似，但是会将右边表（这里的course_info)中的所有记录返回，没有匹配值的左表记录将返回NULL。

```
SELECT student_info.name, course_info.courseName FROM student_info
  RIGHT OUTER JOIN course_info ON (student_info.courseId = course_info.courseId);
```

1.34.4 全外连接

功能描述

根据左表与右表的所有记录进行匹配，没有匹配值的记录返回NULL。

语法格式

```
SELECT attr_expr_list FROM table_reference
  FULL OUTER JOIN table_reference ON join_condition;
```

关键字

FULL OUTER JOIN：根据左表与右表的所有记录进行匹配，没有匹配值的记录返回NULL。

注意事项

所要进行JOIN连接的表必须是已经存在的表，否则会出错。

示例

利用全外连接可以将两张表中的所有记录返回，没有匹配值的左表及右表记录将返回NULL。

```
SELECT student_info.name, course_info.courseName FROM student_info
  FULL OUTER JOIN course_info ON (student_info.courseId = course_info.courseId);
```

1.34.5 隐式连接

功能描述

与内连接功能相同，返回两表中满足WHERE条件的结果集，但不用JOIN显示指定连接条件。

语法格式

```
SELECT table_reference.col_name, table_reference.col_name, ... FROM table_reference, table_reference
  WHERE table_reference.col_name = table_reference.col_name;
```

关键字

WHERE：隐式连接利用WHERE条件实现类似JOIN...ON...的连接，返回匹配的记录。**语法格式**中仅给出等式条件下的WHERE条件过滤，同时也支持不等式WHERE条件过滤。

注意事项

- 所要进行JOIN连接的表必须是已经存在的表，否则会出错。
- 隐式JOIN的命令中不含有JOIN...ON...关键词，而是通过WHERE子句作为连接条件将两张表连接。

示例

返回courseId匹配的学生姓名及课程名称。

```
SELECT student_info.name, course_info.courseName FROM student_info,course_info
  WHERE student_info.courseId = course_info.courseId;
```

1.34.6 笛卡尔连接

功能描述

笛卡尔连接把第一个表的每一条记录和第二个表的所有记录相连接，如果第一个表的记录数为m, 第二个表的记录数为n,则会产生m*n条记录数。

语法格式

```
SELECT attr_expr_list FROM table_reference
  CROSS JOIN table_reference ON join_condition;
```

关键字

join_condition：连接条件，如果该条件恒成立（比如1=1），该连接就是笛卡尔连接。所以，笛卡尔连接输出的记录条数等于被连接表的各记录条数的乘积，若需要进行笛

卡尔积连接，需使用专门的关键词CROSS JOIN。CROSS JOIN是求笛卡尔积的标准方式。

注意事项

所要进行JOIN连接的表必须是已经存在的表，否则会出错。

示例

返回student_info与course_info两张表中学生姓名与课程名称的所有组合。

```
SELECT student_info.name, course_info.courseName FROM student_info
  CROSS JOIN course_info ON (1 = 1);
```

1.34.7 左半连接

功能描述

左半连接用来查看左表中符合JOIN条件的记录。

语法格式

```
SELECT attr_expr_list FROM table_reference
  LEFT SEMI JOIN table_reference ON join_condition;
```

关键字

LEFT SEMI JOIN：只显示左表中的记录。可通过在LEFT SEMI JOIN， WHERE...IN和WHERE EXISTS中嵌套子查询来实现。左半连接与左外连接的区别是，左半连接将返回左表中符合JOIN条件的记录，而左外连接将返回左表所有的记录，匹配不上JOIN条件的记录将返回NULL值。

注意事项

- 所要进行JOIN连接的表必须是已经存在的表，否则会出错。
- 此处的attr_expr_list中所涉及的字段只能是左表中的字段，否则会出错。

示例

返回选课学生的姓名及其所选的课程编号。

```
SELECT student_info.name, student_info.courseId FROM student_info
  LEFT SEMI JOIN course_info ON (student_info.courseId = course_info.courseId);
```

1.34.8 不等值连接

功能描述

不等值连接中，多张表通过不相等的连接值进行连接，并返回满足条件的结果集。

语法格式

```
SELECT attr_expr_list FROM table_reference
  JOIN table reference ON non_equi_join_condition;
```

关键字

non_equi_join_condition：与join_condition类似，只是join条件均为不等式条件。

注意事项

所要进行JOIN连接的表必须是已经存在的表，否则会出错。

示例

返回student_info_1与student_info_2两张表中的所有学生姓名对组合，但不包含相同姓名的姓名对。

```
SELECT student_info_1.name, student_info_2.name FROM student_info_1
  JOIN student_info_2 ON (student_info_1. name <> student_info_2. name);
```

1.35 子查询

1.35.1 WHERE 嵌套子查询

功能描述

在WHERE子句中嵌套子查询，利用子查询的结果作为过滤条件。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference
  WHERE {col_name operator (sub_query) | [NOT] EXISTS sub_query};
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。
- WHERE：WHERE子句嵌套将利用子查询的结果作为过滤条件。
- operator：包含关系运算符中的等式与不等式操作符及IN，NOT IN，EXISTS，NOT EXISTS操作符。
  - 当operator为IN或者NOT IN时，子查询的返回结果必须是单列。
  - 当operator为EXISTS或者NOT EXISTS时，子查询中一定要包含WHERE条件过滤。当子查询中有字段与外部查询相同时，需要在该字段前加上表名。

注意事项

所要查询的表必须是已经存在的表，否则会出错。

示例

先通过子查询在course_info中找到Biology所对应的课程编号，再在student_info表中找到选了该课程编号的学生姓名。

```
SELECT name FROM student_info
  WHERE courseId = (SELECT courseId FROM course_info WHERE courseName = 'Biology');
```

1.35.2 FROM 子句嵌套子查询

功能描述

在FROM子句中嵌套子查询，子查询的结果作为中间过渡表，进而作为外部SELECT语句的数据源。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM (sub_query) [alias];
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- FROM嵌套子查询中，子查询必须要取别名，且别名的命名要早于别名的使用，否则会出错。建议别名不要重名。
- FROM后所跟的子查询结果必须带上前面所取的别名，否则会出错。

示例

返回选了course_info表中课程的学生姓名，并利用DISTINCT关键字进行去重。

```
SELECT DISTINCT name FROM (SELECT name FROM student_info
  JOIN course_info ON student_info.courseId = course_info.courseId) temp;
```

1.35.3 HAVING 子句嵌套子查询

功能描述

在HAVING子句中嵌套子查询，子查询结果将作为HAVING子句的一部分。

语法格式

```
SELECT [ALL | DISTINCT] attr_expr_list FROM table_reference
  GROUP BY groupby_expression
  HAVING aggregate_func(col_name) operator (sub_query);
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。
- groupby_expression：可以是单字段，多字段，也可以是聚合函数，字符串函数等。
- operator：此操作符包含等式操作符与不等式操作符，及IN，NOT IN操作符。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。

- 此处的sub_query与聚合函数的位置不能左右互换。

示例

对表student_info按字段name进行分组，计算每组中记录数，若其记录数等于子查询中表course_info的记录数，返回表student_info中字段name等于表course_info字段name的记录数。

```
SELECT name FROM student_info
  GROUP BY name
  HAVING count(name) = (SELECT count(*) FROM course_info);
```

1.35.4 多层嵌套子查询

功能描述

多层嵌套子查询，即在子查询中嵌套子查询。

语法格式

```
SELECT attr_expr FROM ( SELECT attr_expr FROM ( SELECT attr_expr FROM... ... ) [alias] ) [alias];
```

关键字

- ALL：返回重复的行。为默认选项。其后只能跟*，否则会出错。
- DISTINCT：从结果集移除重复的行。

注意事项

- 所要查询的表必须是已经存在的表，否则会出错。
- 在嵌套查询中必须指定子查询的别名，否则会出错。
- 别名的命名必须在别名的使用之前，否则会出错，建议别名不要重名。

示例

通过三次子查询，最终返回user_info中的name字段。

```
SELECT name FROM ( SELECT name, acc_num FROM ( SELECT name, acc_num, password FROM ( SELECT name, acc_num, password, bank_acc FROM user_info) a ) b ) c;
```

1.36 别名 SELECT

1.36.1 表别名

功能描述

给表或者子查询结果起别名。

语法格式

```
SELECT attr_expr_list FROM table_reference [AS] alias;
```

关键字

- table_reference：可以是表，视图或者子查询。
- AS：可用于连接table_reference和alias，是否添加此关键字不会影响命令执行结果。

注意事项

- 所要查询的表必须是已经存在的，否则会出错。
- 别名的命名必须在别名的使用之前，否则会出错。此外，建议不要重名。

示例

- 给表simple_table起为n的别名，并利用n.name访问simple_table中的name字段。
```
SELECT n.score FROM simple_table n WHERE n.name = "leilei";
```
- 将子查询的结果命令为m，并利用SELECT * FROM m返回子查询中的所有结果。
```
SELECT * FROM (SELECT * FROM simple_table WHERE score > 90) AS m;
```

1.36.2 列别名

功能描述

给列起别名。

语法格式

```
SELECT attr_expr [AS] alias, attr_expr [AS] alias, ... FROM table_reference;
```

关键字

- alias：用于对attr_expr中的字段名称起别名。
- AS：是否添加此关键字不会影响结果。

注意事项

- 所要查询的表必须是已经存在的，否则会出错。
- 别名的命名必须在别名的使用之前，否则会出错。此外，建议不要重名。

示例

先通过子查询SELECT name AS n FROM simple_table WHERE score > 90获得结果，在子查询中给name起的别名n可直接用于外部SELECT语句。

```
SELECT n FROM (SELECT name AS n FROM simple_table WHERE score > 90) m WHERE n = "xiaoming";
```

1.37 集合运算 SELECT

1.37.1 UNION

功能描述

UNION返回多个查询结果的并集。

语法格式

```
select_statement UNION [ALL] select_statement;
```

关键字

UNION：集合运算，以一定条件将表首尾相接，其中每一个SELECT语句返回的列数必须相同，列的类型和列名不一定要相同。

注意事项

- UNION默认是去重的，UNION ALL是不去重的。
- 不能在多个集合运算间（UNION，INTERSECT，EXCEPT）加括号，否则会出错。

示例

返回“SELECT * FROM student _1”查询结果与“SELECT * FROM student _2”查询结果的并集，不包含重复记录。

```
SELECT * FROM student_1 UNION SELECT * FROM student_2;
```

1.37.2 INTERSECT

功能描述

INTERSECT返回多个查询结果的交集。

语法格式

```
select_statement INTERSECT select_statement;
```

关键字

INTERSECT：返回多个查询结果的交集，且每一个SELECT语句返回的列数必须相同，列的类型和列名不一定要相同。INTERSECT默认去重。

注意事项

不能在多个集合运算间（UNION，INTERSECT，EXCEPT）加括号，否则会出错

示例

返回“SELECT * FROM student _1”查询结果与“SELECT * FROM student _2”查询结果的交集，不包含重复记录。

```
SELECT * FROM student _1 INTERSECT SELECT * FROM student _2;
```


1.37.3 EXCEPT

功能描述

返回两个查询结果的差集。

语法格式

```
select_statement EXCEPT select_statement;
```

关键字

EXCEPT：做集合减法。A EXCEPT B将A中所有和B重合的记录扣除，然后返回去重后的A中剩下的记录，EXCEPT默认不去重。与UNION相同，每一个SELECT语句返回的列数必须相同，列的类型和列名不一定要相同。

注意事项

不能在多个集合运算间（UNION，INTERSECT，EXCEPT）加括号，否则会出错

示例

先将“SELECT * FROM student_1”查询结果减去“SELECT * FROM student_2”结果中的重合部分，然后返回剩下来的记录。

```
SELECT * FROM student_1 EXCEPT SELECT * FROM student_2;
```

1.38 WITH...AS

功能描述

通过用WITH...AS定义公共表达式（CTE）来简化查询，提高可阅读性和易维护性。

语法格式

```
WITH cte_name AS (select_statement) sql_containing_cte_name;
```

关键字

- cte_name：公共表达式的名字，不允许重名。
- select_statement：完整的SELECT语句。
- sql_containing_cte_name：包含了刚刚定义的公共表达式的SQL语句

注意事项

- 定义了一个CTE后必须马上使用，否则这个CTE定义将失效。
- 可以通过一次WITH定义多个CTE，中间用逗号连接，后定义的CTE可以引用已经定义的CTE。

示例

将“SELECT courseId FROM course_info WHERE courseName = 'Biology'”定义为公共表达式nv，然后在后续的查询中直接利用nv代替该SELECT语句。

```
WITH nv AS (SELECT courseId FROM course_info WHERE courseName = 'Biology') SELECT DISTINCT courseId FROM nv;
```

1.39 CASE...WHEN

1.39.1 简单 CASE 函数

功能描述

依据input_expression与when_expression的匹配结果跳转到相应的result_expression。

语法格式

```
CASE input_expression WHEN when_expression THEN result_expression [...n] [ELSE else_result_expression] END;
```

关键字

CASE：简单CASE函数中支持子查询，但须注意input_expression与when_expression是可匹配的。

注意事项

如果没有取值为TRUE的input_expression = when_expression，则当指定ELSE子句时，DLI将返回else_result_expression；当没有指定ELSE子句时，返回NULL值。

示例

返回表student中的字段name及与id相匹配的字符。匹配规则如下：

- id为1则返回'a'；
- id为2则返回'b'；
- id为3则返回'c'；
- 否则返回NULL。

```
SELECT name, CASE id WHEN 1 THEN 'a' WHEN 2 THEN 'b' WHEN 3 THEN 'c' ELSE NULL END FROM student;
```

1.39.2 CASE 搜索函数

功能描述

按指定顺序为每个WHEN子句的boolean_expression求值。返回第一个取值为TRUE的boolean_expression的result_expression。

语法格式

```
CASE WHEN boolean_expression THEN result_expression [...n] [ELSE else_result_expression] END;
```

关键字

boolean_expression：可以包含子查询，但整个boolean_expression表达式返回值只能是布尔类型。

注意事项

如果没有取值为TRUE的Boolean_expression，则当指定ELSE子句时，DLI将返回else_result_expression；当没有指定ELSE子句时，返回NULL值。

示例

对表student进行查询，返回字段name及与score对应的结果，score大于等于90返回EXCELLENT，score在（80,90）之间的返回GOOD，否则返回BAD。

```
SELECT name, CASE WHEN score >= 90 THEN 'EXCELLENT' WHEN 80 < score AND score < 90 THEN 'GOOD' ELSE 'BAD' END AS level FROM student;
```

1.40 OVER 子句

功能描述

窗口函数与OVER语句一起使用。OVER语句用于对数据进行分组，并对组内元素进行排序。窗口函数用于给组内的值生成序号。

语法格式

```
SELECT window_func(args) OVER
  ([PARTITION BY col_name, col_name, ...]
   [ORDER BY col_name, col_name, ...]
   [ROWS | RANGE BETWEEN (CURRENT ROW | (UNBOUNDED |[num]) PRECEDING)
  AND (CURRENT ROW | ( UNBOUNDED | [num]) FOLLOWING)]);
```

关键字

- PARTITION BY：可以用一个或多个键分区。和GROUP BY子句类似，PARTITION BY将表按分区键分区，每个分区是一个窗口，窗口函数作用于各个分区。单表分区数最多允许7000个。
- ORDER BY：决定窗口函数求值的顺序。可以用一个或多个键排序。通过ASC或DESC决定升序或降序。窗口由WINDOW子句指定。如果不指定，默认窗口等同于ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW，即窗口从表或分区（如果OVER子句中用PARTITION BY分区）的初始处到当前行。
- WINDOW：通过指定一个行区间来定义窗口。
- CURRENT ROW：表示当前行。
- num PRECEDING：定义窗口的下限，即窗口从当前行向前数num行处开始。
- UNBOUNDED PRECEDING：表示窗口没有下限。
- num FOLLOWING：定义窗口的上限，即窗口从当前行向后数num行处结束。
- UNBOUNDED FOLLOWING：表示窗口没有上限。

- ROWS BETWEEN…和RANGE BETWEEN…的区别：
  - ROW为物理窗口，即根据ORDER BY子句排序后，取前N行及后N行的数据计算（与当前行的值无关，只与排序后的行号相关）。
  - RANGE为逻辑窗口，即指定当前行对应值的范围取值，列数不固定，只要行值在范围内，对应列都包含在内。
- 窗口有以下多种场景，如
  - 窗口只包含当前行。
    ```
    ROWS BETWEEN CURRENT ROW AND CURRENT ROW
    ```
  - 窗口从当前行向前数3行开始，到当前行向后数5行结束。
    ```
    ROWS BETWEEN 3 PRECEDING AND 5 FOLLOWING
    ```
  - 窗口从表或分区的开头开始，到当前行结束。
    ```
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ```
  - 窗口从当前行开始，到表或分区的结尾结束。
    ```
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
    ```
  - 窗口从表或分区的开头开始，到表或分区的结尾结束。
    ```
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
    ```

注意事项

OVER子句包括：PARTITION BY子句、ORDER BY子句和WINDOW子句，可组合使用。OVER子句为空表示窗口为整张表。

示例

上述语句窗口从表或分区的开头开始，到当前行结束，对over_test表按照id字段进行排序，并返回排序好后的id及id所对应的序号。

```
SELECT id, count(id) OVER (ORDER BY id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) FROM over_test;
```

2 Flink Opensource SQL1.12 语法参考

2.1 SQL 语法约束与定义

2.1.1 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。

2.1.2 语法定义

2.1.2.1 DDL 语法定义

2.1.2.1.1 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
```

```
<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。

- 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。

2.1.2.1.2 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
```

2.1.2.1.3 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'
```

2.1.2.2 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
  | '(' ')'
```

```
  | '(' 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 '#'

2.2 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 2-1 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

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

2.3 数据定义语句 DDL

2.3.1 创建源表

2.3.1.1 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)
```



```
)
with (
  'connector' = 'datagen'
);
```

参数说明

表 2-2 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。

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

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。

2.3.1.2 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。

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。

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' = ''
);
```

参数说明

表 2-3 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'。

参数	是否必选	默认值	数据类型	说明
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。
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。
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。

参数	是否必选	默认值	数据类型	说明
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,
  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. 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。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. 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(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。

- 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'
);
```

2.3.1.3 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。
- 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。

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 2-4 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。
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。

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 2-5 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)
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. 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。
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。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. 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. 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,
    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(20210324100000000l,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。重新配置跨源，或者HBase集群安全组放通DLI队列的网段地址。

2.3.1.4 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。

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 2-6 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。
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

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。

数据类型映射

表 2-7 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]

MySQL类型	PostgreSQL类型	Flink SQL类型
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

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. 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。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。

   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,
  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. 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

2.3.1.5 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。

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 2-8 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'。
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'。

参数	是否必选	默认值	数据类型	参数说明
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。
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'。

参数	是否必选	默认值	数据类型	参数说明
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。
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'。

参数	是否必选	默认值	数据类型	参数说明
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。

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 2-9 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。

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. 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。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. 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,
  `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. 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。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. 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,
  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。
  - 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。
  - 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。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。

2.3.1.6 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。
- 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 2-10 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。
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。

参数	是否必选	默认值	数据类型	说明
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。

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. 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。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. 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');

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(20210325120202001,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');
```

2.3.1.7 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 CDC要求Postgre版本为9.6或者10，11，12。
- 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。

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。
- PostgreSQL的版本不能低于PostgreSQL 11。
- 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 2-11 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。
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

参数	是否必选	默认值	数据类型	说明
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。

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. 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。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。
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. 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,
  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。

  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"。

2.3.1.8 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。

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。
  - 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 2-12 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。
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。

参数	是否必选	默认值	数据类型	说明
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。

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. 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。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. 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,
  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
  ```
  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;
```

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})
```

2.3.1.9 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。

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' = '',
 'properties.bootstrap.servers' = '',
 'key.format' = '',
 'value.format' = ''
);
```

参数说明

表 2-13 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。

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。

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：

1. 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。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. 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。

   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(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
+I(202103251505050001,qqShop,2021-03-2515:05:05,500.0,400.0,2021-03-2515:10:00,0003,Cindy,330108)
-U(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
+U(202103251202020001,miniAppShop,2021-03-2512:02:02,60.0,60.0,2021-03-2512:03:00,0002,Bob,330110)
```

常见问题

无

2.3.2 创建结果表

2.3.2.1 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'
);
```

参数说明

表 2-14

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;
```

2.3.2.2 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。

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。

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。
- 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 2-15 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

参数	是否必选	默认值	数据类型	说明
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。
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

参数	是否必选	默认值	数据类型	说明
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. 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。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 │
   │ default_cluster │         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;
   ```

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. 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：

```
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

2.3.2.3 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。

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。

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。

  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'
);
```

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 2-16 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。
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。

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。
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. 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。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. 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
```

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。

- 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。

2.3.2.4 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。

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。

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。

- 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。

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。
- 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。

  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 2-17 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，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。
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。

参数	是否必选	默认值	类型	说明
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。
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。

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. 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。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. 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 (
```



```
 '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"
    }
   },
```

```
    {
      "_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"
      }
    }
  ]
 }
}
```

2.3.2.5 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。
- 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。

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 2-18 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。
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。

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。

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 2-19 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

Flink 数据类型	HBase 转换
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）：

1. 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。
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。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. 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. 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"}

{"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。重新配置跨源，Kafka集群安全组放通DLI队列的网段地址。

2.3.2.6 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。

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' = ''
);
```

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。

参数	是否必选	默认值	类型	说明
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。

数据类型映射

表 2-20 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

MySQL类型	PostgreSQL类型	Flink SQL类型
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]
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. 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。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. 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 (
  '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

2.3.2.7 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。

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。

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 2-21 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。

参数	是否必选	默认参数	数据类型	说明
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。
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'。

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。

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。

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. 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。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. 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,
  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。
  - 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。
  - 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;
```

2.3.2.8 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

打印内容	条件1	条件2
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“1.12”，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 2-22 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

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。

2.3.2.9 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。

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 2-23 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。

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"。

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。 ● 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. 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。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. 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' = '<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"
    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,
 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'
);

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"
```

2.3.2.10 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。

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。
- 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 2-24 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'。
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。

参数	是否必选	默认参数	数据类型	说明
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。

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. 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。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. 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',
  'properties.bootstrap.servers' =  'KafkaAddress1:KafkaPort,KafkAddress2: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

2.3.2.11 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
        ├── 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 2-25 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。

参数	是否必选	默认值	类型	说明
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。

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'
);

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;
```

2.3.3 创建维表

2.3.3.1 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。
- 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。

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 2-26 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。
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。

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。

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

5. 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,
    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。

- 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'
);
```

2.3.3.2 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。
- 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。

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

参数说明

表 2-27 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。

参数	是否必选	默认值	参数类型	说明
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。
lookup.max-retries	否	3	Integer	维表配置，数据拉取最大重试次数。

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 2-28 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，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. 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。
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。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. 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. 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,
     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"}

{"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"}
```

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。重新配置跨源，Kafka集群安全组放通DLI队列的网段地址。

2.3.3.3 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。

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' = ''
);
```

参数说明

表 2-29 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。

参数	是否必选	说明
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。
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。

数据类型映射

表 2-30 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

MySQL类型	PostgreSQL类型	Flink SQL类型
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]
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. 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。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');
```

5. 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
    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":"20210325120202001", "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":"20210325150505001", "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_city_name":"b1","area_county_name":"c2","area_street_name":"d2","region_name":"e1"}

{"order_id":"20210325120202001","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"}

{"order_id":"20210325150505001","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_city_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}
```

常见问题

无。

2.3.3.4 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。

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',
  '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 2-31 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

参数	是否必选	默认值	数据类型	说明
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。
commands-timeout-millis	否	2000	Integer	等待操作完成响应的最大时间。

参数	是否必选	默认值	数据类型	说明
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。

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. 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。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. 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,
  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", "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_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"}

{"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_city_name":"b1","area_county_name":"c3","area_street_name":"d3","region_name":"e1"}

2.3.4 Format

2.3.4.1 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 2-32 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。

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 2-33 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。

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

2.3.4.2 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 2-34 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。

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'
);
```



```
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

2.3.4.3 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 2-35 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。

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"}
```

```
{"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

2.3.4.4 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 2-36

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。

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;
```

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

2.3.4.5 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 2-37

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。

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)
```



```
 ) 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

2.3.4.6 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 2-38

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。

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。

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"}
```

```
{"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

2.3.4.7 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",
```

```
  "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

2.3.4.8 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 2-39

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。

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'。更多细节可查阅字节序。

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

2.4 数据操作语句 DML

2.4.1 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

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

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

```
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;
```

2.4.2 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。
- 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
);
```

2.4.3 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 2-40 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纪录，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 2-41 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。 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;
```

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'
);
```

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 2-42 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

表 2-43 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),
  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 2-44 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;
```

2.4.4 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：

- 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;
```

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;
```

2.4.5 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;
```

2.4.6 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;
```

2.4.7 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;
```

2.5 函数

2.5.1 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。

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。

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.company.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.company.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.company.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

2.5.2 内置函数

2.5.2.1 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2-45。

表 2-45 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。

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

- 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2-46，表中的A和B代表逻辑表达式。

表 2-46 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2-47所示。

表 2-47 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。

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。

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。

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。

2.5.2.2 字符串函数

表 2-48 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."
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."

函数	返回类型	描述
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('abababab', '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"
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"

函数	返回类型	描述
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
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

函数	返回类型	描述
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
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 ` 。

函数	返回类型	描述
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

JSON_VAL 函数使用说明

- 语法

```
STRING JSON_VAL(STRING json_string, STRING json_path)
```

表 2-49 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2-50**。

表 2-50 json_path 参数支持的表达式

表达式	说明
$	根对象

表达式	说明
[]	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",
  "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}
```

2.5.2.3 时间函数

Flink OpenSource SQL所支持的时间函数如**表2-51**所示。

函数说明

表 2-51 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]**"。
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

函数	返回值	描述
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
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。

函数	返回值	描述
(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
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'

函数	返回值	描述
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**

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

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**。
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`
    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

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**
  - 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。

- **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**
  - 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**
  - 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，依次类推。

- **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**

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**

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。

- **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。

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。

参数名	数据类型	参数说明
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

- **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**

  – 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。
- **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。

- **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

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。

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

2.5.2.4 条件函数

函数说明

表 2-52 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

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

2.5.2.5 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。
- 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 2-53 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"

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',
```



```
  '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)
```

2.5.2.6 集合函数

函数说明

表 2-54 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

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

2.5.2.7 值构建函数

函数说明

表 2-55 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)

2.5.2.8 属性访问函数

函数说明

表 2-56 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。

2.5.2.9 Hash 函数

函数说明

表 2-57 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

2.5.2.10 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2-58所示。

表 2-58 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

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

2.5.2.11 表值函数

2.5.2.11.1 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)
```

表 2-59 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：

表 2-60 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 2-61 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

3 Flink Opensource SQL1.10 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
```

```
<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。

- 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。

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

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'
```

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
  | '(' ')'
```

```
  | '(' 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 '#'

3.2 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 3-1 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

语法分类	功能描述
	创建Hbase维表

3.3 数据定义语句 DDL

3.3.1 创建源表

3.3.1.1 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' = ''
);
```

参数说明

表 3-2 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。

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',
```

```
  'connector.properties.bootstrap.servers' = 'xx.xx.xx.xx:9092',
  'connector.properties.group.id' = 'test-group',
  'connector.startup-mode' = 'latest-offset',
  'format.type' = 'json'
);
```

3.3.1.2 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。

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 3-3 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。

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

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' = '',
  'connector.channel' = 'disInput',
  'format.type' = 'csv'
);
```

3.3.1.3 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。

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 3-4 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
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

参数	是否必选	说明
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,
  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'
);
```

3.3.1.4 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。

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。

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' = ''
);
```

参数说明

表 3-5 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}。

参数	是否必选	说明
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.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

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'
);
```

3.3.1.5 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。

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。

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 3-6 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。

参数	是否必选	说明
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。

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'
);
```

3.3.1.6 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。**

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' = ''
);
```

参数说明

表 3-7 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'
);
```

3.3.1.7 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。

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 3-8 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'
);
```

3.3.2 创建结果表

3.3.2.1 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。

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。

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 3-9 参数说明

参数	是否必选	说明
connector.type	是	固定为clickhouse
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,
  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;
```

3.3.2.2 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 3-10 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。

参数	是否必选	说明
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。
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'
);
```

3.3.2.3 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)*
  (',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 3-11 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

参数	是否必选	说明
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

```
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'
);
```

3.3.2.4 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。

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 3-12 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
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' = '',
  'connector.channel' = 'disOutput',
  'connector.partition-key' = 'car_id,car_owner',
  'format.type' = 'csv'
);
```

3.3.2.5 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。

语法格式

```
create table jdbcSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector.type' = 'jdbc',
  'connector.url' = '',
  'connector.table' = '',
  'connector.driver' = '',
  'connector.username' = '',
  'connector.password' = ''
);
```

参数说明

表 3-13 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,
  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'
);
```

3.3.2.6 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。

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。

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' = ''
);
```

参数说明

表 3-14 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。

参数	是否必选	说明
connector.write.escape-string-value	否	是否对string类型值进行转义，默认为false。

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'
);
```

3.3.2.7 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。

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 3-15 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
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

参数	是否必选	说明
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);
```

- 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',
  'connector.data-type' = 'hash'
);

insert into redisSink
  (attr1,attr2,attr3)
  VALUES
  ("car_info","car_id","A1234");
```

3.3.2.8 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。

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 3-16 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。
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' = '',
  'connector.topic-urn' = 'xxxxxx',
  'connector.message-subject' = 'test',
  'connector.message-column' = 'attr1'
);
```

3.3.2.9 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。**

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 3-17 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

参数	是否必选	说明
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

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'
);
```

3.3.2.10 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。

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。

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',
  'connector.version' = '7',
  'connector.hosts' = 'http://xxxx:9200',
  'connector.index' = '',
  'connector.document-type' = '',
  'update-mode' = '',
  'format.type' = 'json'
);
```

参数说明

表 3-18 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

参数	是否必选	说明
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。
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'
);
```

3.3.2.11 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。

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 3-19 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。

参数	是否必选	说明
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。
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' = ''
);
```

3.3.2.12 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>。

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。

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 3-20 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。

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'
);
```

3.3.2.13 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 3-21 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。

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)]
from
  kafkaSource;
```

3.3.2.14 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 3-22 参数说明

参数	是否必选	说明
connector.type	是	固定为filesystem。
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',
  '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;
```

3.3.3 创建维表

3.3.3.1 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' = ''
);
```

参数说明

表 3-23 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

参数	是否必选	说明
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' = '',
  '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' = '',
  '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;
```

3.3.3.2 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 3-24 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

参数	是否必选	说明
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
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' = '',
  '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' = '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' = '',
  '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;
```

3.3.3.3 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。
- **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。**

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' = '',
  'connector.zookeeper.quorum' = ''
);
```

参数说明

表 3-25 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" = "",
  "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'
```

```
);
 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;
```

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

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

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

```
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。
- 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

> **⚠ 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。

表 3-26 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纪录，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 3-27 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。

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：

- 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 3-28 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。	

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 3-29 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;
```

```
// 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。
- 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;
```

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>]
[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 字句

示例

```
SELECT *
FROM Orders
ORDER BY orderTime
LIMIT 3;
```

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;
```

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。

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

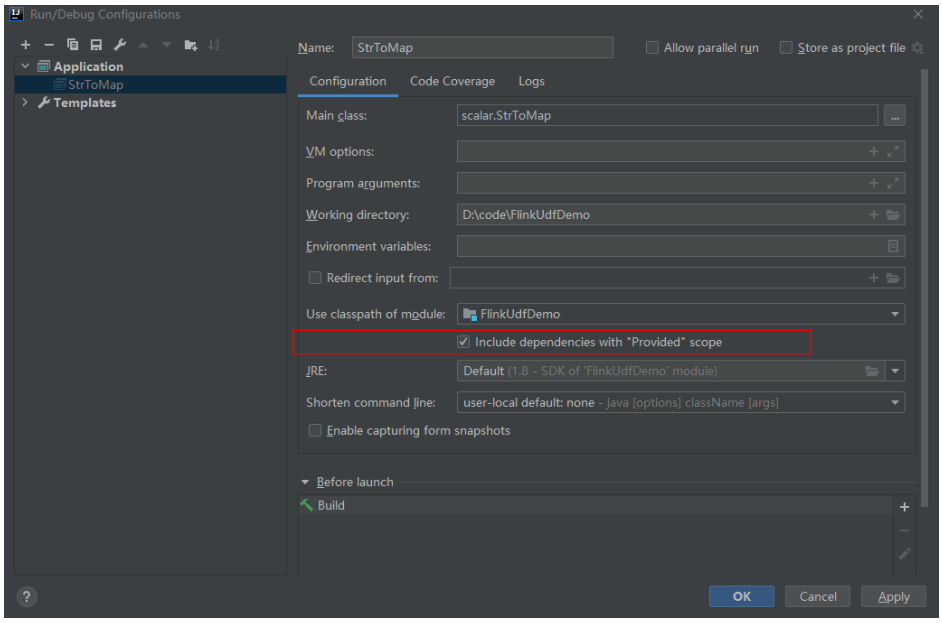
- 暂不支持通过python写UDF、UDTF、UDAF自定义函数。
- 如果使用IntelliJ IDEA工具对创建的自定义函数进行调试，则需要在IDEA上勾选：include dependencies with "Provided" scope，否则本地调试运行时会加载不到pom文件中的依赖包。

  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。



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。

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.company.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.company.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.company.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

3.5.2 内置函数

3.5.2.1 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3-30**。

表 3-30 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。

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

- 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3-31，表中的A和B代表逻辑表达式。

表 3-31 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3-32所示。

表 3-32 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。

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。

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。

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。

3.5.2.2 字符串函数

表 3-33 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
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."

函数	返回类型	描述
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('abababab', '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"
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"

函数	返回类型	描述
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
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

函数	返回类型	描述
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
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

函数	返回类型	描述
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

JSON_VAL 函数使用说明

- 语法

```
STRING JSON_VAL(STRING json_string, STRING json_path)
```

表 3-34 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3-35**。

表 3-35 json_path 参数支持的表达式

表达式	说明
$	根对象

表达式	说明
[]	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',
  '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}
```

3.5.2.3 时间函数

Flink OpenSource SQL所支持的时间函数如表3-36所示。

函数说明

表 3-36 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]**"。
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。

函数	返回值	描述
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
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。

函数	返回值	描述
(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
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'

函数	返回值	描述
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**

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

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**。
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`
    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

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**
  - 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。

- **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**
  - 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**
  - 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，依次类推。

- **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**

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**

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

  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。

- **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

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

- **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

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'。

- **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' 。

- **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])
  ```

- **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

3.5.2.4 条件函数

函数说明

表 3-37 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
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

3.5.2.5 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;
```

表 3-38 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

示例	说明	示例
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。

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;
```

3.5.2.6 集合函数

函数说明

表 3-39 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

3.5.2.7 值构建函数

函数说明

表 3-40 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)

3.5.2.8 属性访问函数

函数说明

表 3-41 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。

值接入函数	函数说明
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。

3.5.2.9 Hash 函数

函数说明

表 3-42 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
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

3.5.2.10 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3-43所示。

表 3-43 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
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

3.5.2.11 表值函数

3.5.2.11.1 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 3-44 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;
```

3.5.2.11.2 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。

语法说明

```
string_split(target, separator)
```

表 3-45 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 3-46 测试源表 disSource 数据和分隔符

target（STRING）	separator （VARCHAR）
test-flink	-

target（STRING）	separator （VARCHAR）
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'
);

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 3-47 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

4 历史版本

4.1 Flink SQL 语法参考

4.1.1 SQL 语法约束与定义

语法约束

- 当前Flink SQL只支持SELECT，FROM，WHERE，UNION，聚合，窗口，流表JOIN以及流流JOIN。
- 数据不能对Source流做insert into操作。
- Sink流不能用来做查询操作。

语法支持范围

- 基础类型： VARCHAR，STRING，BOOLEAN，TINYINT，SMALLINT，INTEGER/INT，BIGINT，REAL/FLOAT，DOUBLE，DECIMAL，DATE，TIME，TIMESTAMP
- Array：使用[]进行引用。例如：
  ```
  insert into temp select CARDINALITY(ARRAY[1,2,3]) FROM OrderA;
  ```

语法定义

```
INSERT INTO stream_name query;
query:
  values
  | {
      select
      | selectWithoutFrom
      | query UNION [ ALL ] query
    }

orderItem:
  expression [ ASC | DESC ]

select:
  SELECT
  { * | projectItem [, projectItem ]* }
  FROM tableExpression [ JOIN tableExpression ]
  [ WHERE booleanExpression ]
```

```
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]

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

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

tableExpression:
  tableReference

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

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

values:
  VALUES expression [, expression ]*

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

4.1.2 流作业 SQL 语法概览

本章节介绍了目前DLI所提供的Flink 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。

表 4-1 流作业语法概览

语法分类	功能描述
创建输入流	**DIS输入流**
	DMS输入流
创建输入流	**MRS Kafka输入流**
	开源Kafka输入流
	OBS输入流
创建输出流	**CSS Elasticsearch输出流**
	DCS输出流
	DDS输出流
	DIS输出流
	DMS输出流
	DWS输出流（通过JDBC方式）

语法分类	功能描述
	DWS输出流（通过OBS转储方式）
创建输出流	MRS HBase输出流
	MRS Kafka输出流
	开源Kafka输出流
	OBS输出流
	RDS输出流
创建输出流	SMN输出流
	文件系统输出流(推荐）
创建中间流	创建中间流
创建维表	创建Redis表
	创建RDS表
自拓展生态	自拓展输入流
	自拓展输出流

4.1.3 创建输入流

4.1.3.1 DIS 输入流

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。

语法格式

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "dis",
    region = "",
    channel = "",
    partition_count = "",
    encode = "",
    field_delimiter = "",
    offset= "");
```

关键字

表 4-2 关键字说明

参数	是否必选	说明
type	是	数据源类型，“dis”表示数据源为数据接入服务。
region	是	数据所在的DIS区域。
ak	否	访问密钥ID(Access Key ID)。
sk	否	Secret Access Key，与访问密钥ID结合使用的密钥。
channel	是	数据所在的DIS通道名称。
partition_count	否	数据所在的DIS通道分区数。该参数和partition_range参数不能同时配置。当该参数没有配置的时候默认读取所有partition。
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。当该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。
encode	是	数据编码格式，可选为“csv”、“json”、“xml”、“email”、“blob”和“user_defined”。 ● 若编码格式为“csv”，则需配置“field_delimiter”属性。 ● 若编码格式为“json”，则需配置“json_config”属性。 ● 若编码格式为“xml”，则需配置“xml_config”属性。 ● 若编码格式为“email”，则需配置“email_key”属性。 ● 若编码格式为“blob”，表示不对接收的数据进行解析，流属性仅能有一个且数据格式为ARRAY[TINYINT]。 ● 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
field_delimiter	否	属性分隔符，仅当编码格式为csv时该参数需要填写，例如配置为“,”。
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 ● 当引用符号为双引号时，请设置quote = "\u005c\u0022"进行转义。 ● 当引用符号为单引号时，则设置quote = "'"。 **说明** ● 目前仅适用于CSV格式。 ● 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。

参数	是否必选	说明
json_config	否	当编码格式为json时，用户需要通过该参数来指定json字段和流定义字段的映射关系，格式为“field1=data_json.field1; field2=data_json.field2; field3=$”，其中field3=$表示field3的内容为整个json串。
xml_config	否	当编码格式为xml时，用户需要通过该参数来指定xml字段和流定义字段的映射关系，格式为“field1=data_xml.field1; field2=data_xml.field2”。
email_key	否	当编码格式为email时，用户需要通过该参数来指定需要提取的信息，需要列出信息的key值，需要与流定义字段一一对应，多个key值时以逗号分隔，例如“Message-ID, Date, Subject, body”，其中由于邮件正文没有关键字，DLI规定其关键字为“body”。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现解码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现解码类的入参，仅支持一个string类型的参数。
offset	否	• 当启动作业后再获取数据，则该参数无效。 • 当获取数据后再启动作业，用户可以根据需求设置该参数的数值。 例如当offset= "100"时，则表示DLI从DIS服务中的第100条数据开始处理。
start_time	否	DIS数据读取起始时间。 • 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。 • 当没有配置start_time但配置了offset的时候，则从offset开始读取数据。
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。
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。
checkpoint_interval	否	DIS源算子做checkpoint的时间间隔，单位秒，默认为60。

注意事项

在创建Source Stream时可以指定时间模型以便在后续计算中使用，当前DLI支持Processing Time和Event Time两种时间模型，具体使用语法可以参考**配置时间模型**。

示例

- CSV编码格式：从DIS通道读取数据，记录为csv编码，并且以逗号为分隔符。

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT,
  car_timestamp LONG
)
  WITH (
    type = "dis",
    region = "xxx",
    channel = "dliinput",
    encode = "csv",
    field_delimiter = ","
);
```

- JSON编码格式：从DIS通道读取数据，记录为json编码。数据示例：{"car": {"car_id":"ZJA710XC", "car_owner":"coco", "car_age":5, "average_speed":80, "total_miles":15000, "car_timestamp":1526438880}}。

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT,
  car_timestamp LONG
)
  WITH (
    type = "dis",
    region = "xxx",
    channel = "dliinput",
    encode = "json",
    json_config = "car_id=car.car_id;car_owner =car.car_owner;car_age=car.car_age;average_speed
=car.average_speed ;total_miles=car.total_miles;"
);
```

- XML编码格式：从DIS通道读取数据，记录为xml编码。

```
CREATE SOURCE STREAM person_infos (
    pid BIGINT,
    pname STRING,
    page int,
    plocation STRING,
    pbir DATE,
    phealthy BOOLEAN,
    pgrade ARRAY[STRING]
)
  WITH (
    type = "dis",
    region = "xxx",
    channel = "dis-dli-input",
    encode = "xml",
    field_delimiter = ",",
    xml_config =
"pid=person.pid;page=person.page;pname=person.pname;plocation=person.plocation;pbir=person.pbir;
pgrade=person.pgrade;phealthy=person.phealthy"
);
```

  xml数据示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
```



```
<root>
  <person>
    <pid>362305199010025042</pid>
    <pname>xiaoming</pname>
    <page>28</page>
    <plocation>xxx</plocation>
    <pbir>1990-10-02</pbir>
    <phealthy>true</phealthy>
    <pgrade>[A,B,C]</pgrade>
  </person>
</root>
```

- EMAIL编码格式：从DIS通道读取数据，每条记录为一封完整邮件。

```
CREATE SOURCE STREAM email_infos (
  Event_ID String,
  Event_Time Date,
  Subject String,
  From_Email String,
  To_EMAIL String,
  CC_EMAIL Array[String],
  BCC_EMAIL String,
  MessageBody String,
  Mime_Version String,
  Content_Type String,
  charset String,
  Content_Transfer_Encoding String
)
  WITH (
    type = "dis",
    region = "xxx",
    channel = "dliinput",
    encode = "email",
    email_key = "Message-ID, Date, Subject, From, To, CC, BCC, Body, Mime-Version, Content-Type,
charset, Content_Transfer_Encoding"
);
```

email数据示例如下：

```
Message-ID: <20090629183903250425 4@sample.com>
Date: Fri, 11 May 2001 09:54:00 -0700 (PDT)
From: user1@sample.com
To: user2@sample.com, user3@sample.com
Subject:  "Hello World"
Cc: user4@sample.com, user5@sample.com
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Bcc: user6@sample.com, user7@sample.com
X-From: user1
X-To: user2, user3
X-cc: user4, user5
X-bcc:
X-Folder: \user2_June2001\Notes Folders\Notes inbox
X-Origin: user8
X-FileName: sample.nsf

Dear Associate / Analyst Committee:

Hello World!

Thank you,

Associate / Analyst Program
user1
```


4.1.3.2 DMS 输入流

分布式消息服务（Distributed Message Service，简称DMS）是一项基于高可用分布式集群技术的消息中间件服务，提供了可靠且可扩展的托管消息队列，用于收发消息和存储消息。

分布式消息服务Kafka是一款基于开源社区版Kafka提供的消息队列服务，向用户提供可靠的全托管式的Kafka消息队列。

DLI支持创建输入流从DMS的Kafka获取数据，作为作业的输入数据。

创建DMS Kafka输入流的语法与创建开源Apache Kafka输入流一样，具体请参见**开源Kafka输入流**。

4.1.3.3 MRS Kafka 输入流

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。MRS基于Apache Kafka在平台部署并托管了Kafka集群。

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。

语法格式

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "kafka",
    kafka_bootstrap_servers = "",
    kafka_group_id = "",
    kafka_topic = "",
    encode = "json"
  );
```

关键字

表 4-3 关键字说明

参数	是否必选	说明
type	是	数据源类型，“Kafka”表示数据源。
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过增强型跨源开通DLI队列和Kafka集群的连接）。
kafka_group_id	否	group id。
kafka_topic	是	读取的Kafka的topic。目前只支持读取单个topic。

参数	是否必选	说明
encode	是	数据编码格式，可选为“csv”、“json”、“blob”和“user_defined”。 • 若编码格式为“csv”，则需配置“field_delimiter”属性。 • 若编码格式为“json”，则需配置“json_config”属性。 • 当编码格式为"blob"时，表示不对接收的数据进行解析，流属性仅能有一个且为Array[TINYINT]类型。 • 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现解码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现解码类的入参，仅支持一个string类型的参数。
krb_auth	否	创建跨源认证的认证名。开启kerberos认证时，需配置该参数。 **说明** 请确保在DLI队列host文件中添加MRS集群master节点的“/etc/hosts”信息。
json_config	否	当encode为json时，用户可以通过该参数指定json字段和流属性字段的映射关系。 格式："field1=json_field1;field2=json_field2" 格式说明：field1、field2为创建的表字段名称。json_field1、json_field2为kafka输入数据json串的key字段名称。 具体使用方法可以参考**示例**说明。
field_delimiter	否	当encode为csv时，用于指定csv字段分隔符，默认为逗号。
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 • 当引用符号为双引号时，请设置quote = "\u005c\u0022"进行转义。 • 当引用符号为单引号时，则设置quote = "'"。 **说明** • 目前仅适用于CSV格式。 • 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。

参数	是否必选	说明
start_time	否	kafka数据读取起始时间。 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。start_time要不大于当前时间，若大于当前时间，则不会有数据读取出。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"
kafka_certificate_name	否	跨源认证信息名称。跨源认证信息类型为“Kafka_SSL”时，该参数有效。 **说明** • 指定该配置项时，服务仅加载该认证下指定的文件和密码，系统将自动设置到“kafka_properties”属性中。 • Kafka SSL认证需要的其他配置信息，需要用户手动在“kafka_properties”属性中配置。

注意事项

在创建Source Stream时可以指定时间模型以便在后续计算中使用，当前DLI支持Processing Time和Event Time两种时间模型，具体使用语法可以参考**配置时间模型**。

示例

- 从Kafka名称为test的topic中读取数据。

```
CREATE SOURCE STREAM kafka_source (
  name STRING,
  age int
 )
  WITH (
    type = "kafka",
    kafka_bootstrap_servers = "ip1:port1,ip2:port2",
    kafka_group_id = "sourcegroup1",
    kafka_topic = "test",
    encode = "json"
);
```

- 从Kafka读取对象为test的topic，使用json_config将json数据和表字段对应。

  数据编码格式为json且不含嵌套，例如：

```
{"attr1": "lilei", "attr2": 18}
```

  建表语句参考如下：

```
CREATE SOURCE STREAM kafka_source (name STRING, age int)
WITH (
  type = "kafka",
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",
  kafka_group_id = "sourcegroup1",
  kafka_topic = "test",
  encode = "json",
  json_config = "name=attr1;age=attr2"
);
```

4.1.3.4 开源 Kafka 输入流

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。

语法格式

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "kafka",
    kafka_bootstrap_servers = "",
    kafka_group_id = "",
    kafka_topic = "",
    encode = "json",
    json_config=""
  );
```

关键字

表 4-4 关键字说明

参数	是否必选	说明
type	是	数据源类型，“Kafka”表示数据源。
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过增强型跨源开通DLI队列和Kafka集群的连接）。
kafka_group_id	否	group id。
kafka_topic	是	读取的Kafka的topic。目前只支持读取单个topic。
encode	是	数据编码格式，可选为“csv”、“json”、“blob”和“user_defined”。 ● 若编码格式为“csv”，则需配置“field_delimiter”属性。 ● 若编码格式为“json”，则需配置“json_config”属性。 ● 当编码格式为"blob"时，表示不对接收的数据进行解析，当前表仅能有一个且为Array[TINYINT]类型的表字段。 ● 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。

参数	是否必选	说明
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现解码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现解码类的入参，仅支持一个string类型的参数。
json_config	否	当encode为json时，用户可以通过该参数指定json字段和流属性字段的映射关系。 格式："field1=json_field1;field2=json_field2" 格式说明：field1、field2为创建的表字段名称。json_field1、json_field2为kafka输入数据json串的key字段名称。 具体使用方法可以参考**示例**说明。 **说明** 如果定义的source stream中的属性和json中的属性名称相同，json_configs可以不用配置。
field_delimiter	否	当encode为csv时，用于指定csv字段分隔符，默认为逗号。
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 • 当引用符号为双引号时，请设置quote = "\u005c\u0022"进行转义。 • 当引用符号为单引号时，则设置quote = "'"。 **说明** • 目前仅适用于CSV格式。 • 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。
start_time	否	kafka数据读取起始时间。 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。start_time要不大于当前时间，若大于当前时间，则不会有数据读取出。 该参数配置后，只会读取Kafka topic在该时间点后产生的数据。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"。具体的属性值可以参考**Apache Kafka**中的描述。

参数	是否必选	说明
kafka_certificate_name	否	跨源认证信息名称。跨源认证信息类型为“Kafka_SSL”时，该参数有效。 **说明** • 指定该配置项时，服务仅加载该认证下指定的文件和密码，系统将自动设置到“kafka_properties”属性中。 • Kafka SSL认证需要的其他配置信息，需要用户手动在“kafka_properties”属性中配置。

注意事项

在创建Source Stream时可以指定时间模型以便在后续计算中使用，当前DLI支持Processing Time和Event Time两种时间模型，具体使用语法可以参考**配置时间模型**。

示例

- 从Kafka读取对象为test的topic。数据编码格式为json且不含嵌套，例如：{"attr1": "lilei", "attr2": 18}。

```
CREATE SOURCE STREAM kafka_source (name STRING, age int)
WITH (
  type = "kafka",
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",
  kafka_group_id = "sourcegroup1",
  kafka_topic = "test",
  encode = "json",
  json_config = "name=attr1;age=attr2"
);
```

- 从Kafka读取对象为test的topic。数据编码格式为json且包含嵌套。本示例使用了复杂数据类型ROW，ROW使用语法可以参考**数据类型**。

  测试数据参考如下：

```
{
    "id":"1",
    "type2":"online",
    "data":{
        "patient_id":1234,
        "name":"bob1234"
    }
}
```

  则对应建表语句示例为：

```
CREATE SOURCE STREAM kafka_source
(
  id STRING,
  type2 STRING,
  data ROW<
    patient_id STRING,
    name STRING>
)
WITH (
  type = "kafka",
  kafka_bootstrap_servers = "ip1:port1,ip2:port2",
  kafka_group_id = "sourcegroup1",
  kafka_topic = "test",
  encode = "json"
);

CREATE SINK STREAM kafka_sink
```

```
(
  id STRING,
  type2 STRING,
  patient_id STRING,
  name STRING
)
  WITH (
    type="kafka",
    kafka_bootstrap_servers =  "ip1:port1,ip2:port2",
    kafka_topic = "testsink",
    encode = "csv"
  );

INSERT INTO kafka_sink select id, type2, data.patient_id, data.name from kafka_source;
```

4.1.3.5 OBS 输入流

功能描述

创建source流从对象存储服务（OBS）获取数据。DLI从OBS上读取用户存储的数据，作为作业的输入数据。适用于大数据分析、原生云应用程序数据、静态网站托管、备份/活跃归档、深度/冷归档等场景。

对象存储服务（Object Storage Service，简称OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。OBS的更多信息，请参见。

语法格式

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "obs",
    region = "",
    bucket = "",
    object_name = "",
    row_delimiter = "\n",
    field_delimiter = '',
    version_id = ""
  );
```

关键字

表 4-5 关键字说明

参数	是否必选	说明
type	是	数据源类型，“obs”表示数据源为对象存储服务。
region	是	对象存储服务所在区域。
encode	否	数据的编码格式，可以为“csv”或者“json”。默认值为“csv”。
ak	否	访问密钥ID(Access Key ID)。
sk	否	Secret Access Key，与访问密钥ID结合使用的密钥。
bucket	是	数据所在的OBS桶名。

参数	是否必选	说明
object_name	是	数据所在OBS桶中的对象名。如果对象不在OBS根目录下，则需添加文件夹名，例如：test/test.csv。对象文件格式参考“encode”参数。
row_delimiter	是	行间的分隔符。
field_delimiter	否	属性分隔符。 • 当“encode”参数为csv时，该参数必选。用户可以自定义属性分隔符。 • 当“encode”参数为json时，该参数不需要填写。
quote	否	可以指定数据格式中的引用符号，在两个引用符号之间的属性分隔符会被当做普通字符处理。 • 当引用符号为双引号时，请设置quote = "\u005c\u0022"进行转义。 • 当引用符号为单引号时，则设置quote = "'"。 **说明** • 目前只适用于CSV格式。 • 设置引用符号后，必须保证每个字段中包含0个或者偶数个引用符号，否则会解析失败。
version_id	否	版本号，当obs里的桶或对象有设置版本的时候需填写，否则不用配置该项。

注意事项

在创建Source Stream时可以指定时间模型以便在后续计算中使用，当前DLI支持Processing Time和Event Time两种时间模型，具体使用语法可以参考**配置时间模型**。

示例

- 从OBS的桶读取对象为input.csv的文件，文件以'\n'划行， 以','划列。

  测试输入数据input.csv可以先通过新建input.txt复制如下文本数据，再另存为input.csv格式文件。将input.csv上传到对应OBS桶目录下。例如，当前上传到：“dli-test-obs01”桶目录下。

  ```
  1,2,3,4,1403149534
  5,6,7,8,1403149535
  ```

  创建表参考如下：

  ```
  CREATE SOURCE STREAM car_infos (
    car_id STRING,
    car_owner STRING,
    car_brand STRING,
    car_price INT,
    car_timestamp LONG
  )
    WITH (
      type = "obs",
      bucket = "dli-test-obs01",
      region = "xxx",
  ```

```
  object_name = "input.csv",
  row_delimiter = "\n",
  field_delimiter = ","
);
```

- 从OBS的桶读取对象为input.json的文件，文件以'\n'划行。

```
CREATE SOURCE STREAM obs_source (
 str STRING
)
 WITH (
  type = "obs",
  bucket = "obssource",
  region = "xxx",
  encode = "json",
  row_delimiter = "\n",
  object_name = "input.json"
);
```

4.1.4 创建输出流

4.1.4.1 MRS OpenTSDB 输出流

功能描述

DLI将Flink作业的输出数据输出到MRS的OpenTSDB中。

前提条件

- 确保MRS的集群已经安装了OpenTSDB。
- 该场景作业需要运行在DLI的独享队列上，因此要与MRS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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
 WITH (
  type = "opentsdb",
  region = "",
  tsdb_metrics = "",
  tsdb_timestamps = "",
  tsdb_values = "",
  tsdb_tags = "",
  batch_insert_data_num = ""
 )
```

关键字

表 4-6 关键字说明

参数	是否必选	说明
type	是	输出通道类型，“opentsdb”表示输出到MRS的OpenTSDB。
region	是	MRS服务所在区域。

参数	是否必选	说明
tsdb_link_address	是	MRS中OpenTSDB实例的服务地址，格式为http://ip:port或者https://ip:port。 **说明** 配置项tsd.https.enabled为true时，需要使用https，注意https暂时不支持证书认证。
tsdb_metrics	是	数据点的metric，支持参数化。
tsdb_timestamps	是	数据点的timestamp，数据类型支持LONG、INT、SHORT和STRING，仅支持指定动态列。
tsdb_values	是	数据点的value，数据类型支持SHORT、INT、LONG、FLOAT、DOUBLE和STRING，支持指定动态列或者常数值。
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，支持参数化。
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，上限为65536，默认值为8。

注意事项

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。

示例

将流weather_out的数据输出到MRS服务的OpenTSDB中。

```
CREATE SINK STREAM weather_out (
  timestamp_value LONG, /* 时间 */
  temperature FLOAT, /* 温度值 */
  humidity FLOAT, /* 湿度值 */
  location STRING /* 地点 */
)
  WITH (
    type = "opentsdb",
    region = "xxx",
    tsdb_link_address = "https://x.x.x.x:4242",
    tsdb_metrics = "weather",
    tsdb_timestamps = "${timestamp_value}",
    tsdb_values = "${temperature}; ${humidity}",
    tsdb_tags = "location:${location},signify:temperature; location:${location},signify:humidity",
    batch_insert_data_num = "10"
);
```

4.1.4.2 CSS Elasticsearch 输出流

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。

说明

创建CSS集群时如果开启了安全模式，后续将无法关闭。

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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "es",
    region = "",
    cluster_address = "",
    es_index = "",
    es_type= "",
    es_fields= "",
    batch_insert_data_num= ""
  );
```

关键字

表 4-7 关键字说明

参数	是否必选	说明
type	是	输出通道类型，es表示输出到云搜索服务中。
region	是	数据所在的云搜索服务所在区域。
cluster_address	是	云搜索服务集群的内网访问地址，例如：x.x.x.x:x，多个地址时以逗号分隔。
es_index	是	待插入数据的索引，支持参数化。对应CSS服务中的index。
es_type	是	待插入数据的文档类型，支持参数化。对应CSS服务中的type。 若使用的es版本为6.x，则该值不能以"_"开头。 若使用的es版本为7.x，如果提前预置CSS服务中的type，则es_type需为"_doc"，否则可为符合CSS规范的值。
es_fields	是	待插入数据字段的key，具体形式如："id,f1,f2,f3,f4"，并且保证与sink中数据列一一对应；如果不使用key，而是采用随机的属性字段，则无需使用id关键字，具体形式如："f1,f2,f3,f4,f5"。对应CSS服务中的filed。

参数	是否必选	说明
batch_insert_data_num	是	表示一次性批量写入的数据量，值必须为正整数，单位为：条。上限为65536，默认值为10。
action	否	当值为add时，表示遇到相同id时，数据被强制覆盖，当值为upsert时，表示遇到相同id时，更新数据（选择upsert时，es_fields字段中必须指定id），默认值为add。
enable_output_null	否	使用该参数来配置是否输出空字段。当该参数为true表示输出空字段（值为null），若为false表示不输出空字段。默认为false。
max_record_num_cache	否	记录最大缓存数。
es_certificate_name	否	跨源认证信息名称。 若es集群开启安全模式且开启https，则使用证书进行访问，创建的跨源认证类型需要为“CSS”。 若es集群开启安全模式，但关闭https，则使用证书和账号密码进行访问，创建的跨源认证类型需要为"Password"。

注意事项

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。

示例

将流qualified_cars的数据输出到云搜索服务的集群。

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT
)
  WITH (
    type = "es",
    region = "xxx",
    cluster_address = "192.168.0.212:9200",
    es_index = "car",
    es_type = "information",
    es_fields = "id,owner,age,speed,miles",
    batch_insert_data_num = "10"
);
```

4.1.4.3 DCS 输出流

功能描述

DLI将Flink作业的输出数据输出到分布式缓存服务（DCS）的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/**。

分布式缓存服务（DCS）为DLI提供兼容Redis的即开即用、安全可靠、弹性扩容、便捷管理的在线分布式缓存能力，满足用户高并发及快速数据访问的业务诉求。

前提条件

- 请务必确保您的账户下已在分布式缓存服务（DCS）里创建了Redis类型的缓存实例。
- 该场景作业需要运行在DLI的独享队列上，因此要与DCS集群建立跨源连接，且用户可以根据实际所需设置相应安全组规则。
- 用户通过VPC对等访问DCS实例时，除了满足VPC对等网跨VPC访问的约束之外，还存在如下约束：
  - 当创建DCS实例时使用了172.16.0.0/12~24网段时，DLI队列不能在192.168.1.0/24、192.168.2.0/24、192.168.3.0/24网段。
  - 当创建DCS实例时使用了192.168.0.0/16~24网段时，DLI队列不能在172.31.1.0/24、172.31.2.0/24、172.31.3.0/24网段。
  - 当创建DCS实例时使用了10.0.0.0/8~24网段时，DLI队列不能在172.31.1.0/24、172.31.2.0/24、172.31.3.0/24网段。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "dcs_redis",
    region = "",
    cluster_address = "",
    password = "",
    value_type= "",key_value= ""
  );
```

关键字

表 4-8 关键字说明

参数	是否必选	说明
type	是	输出通道类型，dcs_redis表示输出到分布式缓存服务的Redis存储系统中。
region	是	数据所在的DCS所在区域。
cluster_address	是	Redis实例连接地址。

参数	是否必选	说明
password	否	Redis实例连接密码，当设置为免密访问时，省略该配置项。
value_type	是	该参数可配置为如下选项或选项的组合： ● 支持指定插入数据类型，包括：string, list, hash, set, zset； ● 支持设置key的过期时间，包括expire, pexpire, expireAt, pexpireAt； ● 支持删除key命令，包括del, hdel； 当需要使用多个命令时，用“;”分隔。
key_value	是	设置具体的key和value，key_value对必须与value_type所指定的类型数相对应，用“;”分隔，且key和value均支持参数化，动态列名采用${列名}表示。

注意事项

- 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_${car_brand}时，如果一条记录的car_brand列值为BMW，则该配置项在该条记录下为car_BMW。
- 字符":", ",", ";", "$", "{", "}"已被征用为特殊分隔符，暂时没有提供转义功能，禁止在key和value中作为普通字符使用，否则会影响解析，导致程序异常。

示例

将流qualified_cars的数据输出到DCS服务的Redis类型的缓存实例中。

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed DOUBLE,
  total_miles DOUBLE
)
  WITH (
    type = "dcs_redis",
    cluster_address = "192.168.0.34:6379",
    password = "xxxxxxxx",
    value_type = "string; list; hash; set; zset",
    key_value = "${car_id}_str: ${car_owner}; name_list: ${car_owner}; ${car_id}_hash: {name:${car_owner}, age: ${car_age}}; name_set:   ${car_owner}; math_zset: {${car_owner}:${average_speed}}"
  );
```

4.1.4.4 DDS 输出流

功能描述

DLI将作业的输出数据输出到文档数据库服务（DDS）中。

文档数据库服务（Document Database Service，简称DDS）完全兼容MongoDB协议，提供安全、高可用、高可靠、弹性伸缩和易用的数据库服务，同时提供一键部署、弹性扩容、容灾、备份、恢复、监控和告警等功能。

前提条件

- 请务必确保您的账户下已在文档数据库服务（DDS）里创建了DDS实例。
  如何创建DDS实例，请参考中“快速购买文档数据库实例”章节。
- 目前仅支持未开启SSL认证的集群实例，不支持副本集与单节点的类型实例。
- 该场景作业需要运行在DLI的独享队列上，请确保已创建DLI独享队列。
- 确保DLI独享队列与DDS集群建立跨源连接，且用户可以根据实际所需设置相应安全组规则。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "dds",
    username = "",
    password = "",
    db_url = "",
    field_names = ""
  );
```

关键字

表 4-9 关键字说明

参数	是否必选	说明
type	是	输出通道类型，dds表示输出到文档数据库服务中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	DDS实例的访问地址，形如：ip1:port,ip2:port/database/collection。
field_names	是	待插入数据字段的key，具体形式如："f1,f2,f3"，并且保证与sink中数据列一一对应。
batch_insert_data_num	否	表示一次性批量写入的数据量，值必须为正整数，默认值为10。

示例

将流qualified_cars 的数据输出到文档数据库collectionTest。

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT
)
  WITH (
    type = "dds",
    region = "xxx",
    db_url = "192.168.0.8:8635,192.168.0.130:8635/dbtest/collectionTest",
```

```
  username = "xxxxxxxxxx",
  password =  "xxxxxxxxxx",
  field_names = "car_id,car_owner,car_age,average_speed,total_miles",
  batch_insert_data_num = "10"
);
```

4.1.4.5 DIS 输出流

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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "dis",
    region = "",
    channel = "",
    partition_key = "",
    encode= "",
    field_delimiter= ""
  );
```

关键字

表 4-10 关键字说明

参数	是否必选	说明
type	是	输出通道类型，dis表示输出到数据接入服务。
region	是	数据所在的DIS所在区域。
ak	否	访问密钥ID(Access Key ID)。
sk	否	Secret Access Key，与访问密钥ID结合使用的密钥。
channel	是	DIS通道。
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。

参数	是否必选	说明
encode	是	数据编码格式，可选为“csv”、“json”和“user_defined”。 **说明** • 若编码格式为“csv”，则需配置“field_delimiter”属性。 • 若编码格式为“json”，则需使用“enable_output_null”来配置是否输出空字段，具体见示例。 • 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
field_delimiter	是	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，如：“，”。 • 当编码格式为json时，则不需要设置属性之间的分隔符。
json_config	否	当编码格式为json时，用户可以通过该参数来指定json字段和流定义字段的映射关系，格式为“field1=data_json.field1;field2=data_json.field2”。
enable_output_null	否	当编码格式为json时，需使用该参数来配置是否输出空字段。 当该参数为“true”表示输出空字段（值为null），若为“false”表示不输出空字段。默认值为“true”。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现编码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现编码类的入参，仅支持一个string类型的参数。

注意事项

无。

示例

- CSV编码格式：数据输出到DIS通道，使用csv编码，并且以逗号为分隔符，多个分区用car_owner做为key进行分发。数据输出示例："ZJA710XC", "lilei", "BMW", 700000。

```
CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
```

```
  car_price INT
)
  WITH (
    type = "dis",
    region = "xxx",
    channel = "dlioutput",
    encode = "csv",
    field_delimiter = ","
);
```

- JSON编码格式：数据输出到DIS通道，使用json编码，多个分区用car_owner，car_brand 做为key进行分发,“enableOutputNull”为“true”表示输出空字段（值为null），若为“false”表示不输出空字段。数据示例："car_id ":"ZJA710XC", "car_owner ":"lilei", "car_brand ":"BMW", "car_price ":700000。

```
CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
    type = "dis",
    channel = "dlioutput",
    region = "xxx",
    partition_key = "car_owner,car_brand",
    encode = "json",
    enable_output_null = "false"
);
```

4.1.4.6 DMS 输出流

分布式消息服务Kafka是一款基于开源社区版Kafka提供的消息队列服务，向用户提供可靠的全托管式的Kafka消息队列。

DLI支持将作业的输出数据输出到DMS的Kafka实例中。

创建DMS Kafka输出流的语法与创建开源Apache Kafka输出流一样，具体请参见**MRS Kafka输出流**。

4.1.4.7 DWS 输出流（通过 JDBC 方式）

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。

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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "rds",
    username = "",
    password = "",
    db_url = "",
    table_name = ""
  );
```

关键字

表 4-11 关键字说明

参数	是否必选	说明
type	是	输出通道类型，rds表示输出到关系型数据库或者数据仓库服务中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址格式为：postgresql://ip:port/database。
table_name	是	要插入数据的数据库表名。数据库表需事先创建好。

db_columns	否	支持配置输出流属性和数据库表属性的对应关系，需严格按照输出流的属性顺序配置。
		示例：
		`create sink stream a3(student_name  string, student_age int)`
		`  with (`
		`    type = "rds",`
		`    username = "root",`
		`    password = "xxxxxxxx",`
		`    db_url = "postgresql://192.168.0.102:8000/test1",`
		`    db_columns = "name,age",`
		`    table_name = "t1"`
		`  );`
		student_name对应数据库里的name属性，student_age对应数据库里的age属性。
		说明
		• 当不配置db_columns时，若输出流属性个数小于数据库表属性个数，并且数据库多出的属性都是nullable或者有默认值时，这种情况也允许。

参数	是否必选	说明
primary_key	否	如果想通过主键实时更新表中的数据，需要在创建数据表的时候增加primary_key配置项，如下面例子中的c_timeminute。配置primary_key后，在进行数据写入操作时，如果primary_key存在，则进行更新操作，否则进行插入操作。 示例： CREATE SINK STREAM test(c_timeminute LONG, c_cnt LONG) WITH (type = "rds", username = "root", password = "xxxxxxxx", db_url = "postgresql://192.168.0.12:8000/test", table_name = "test", primary_key = "c_timeminute");

注意事项

stream_id所定义的流格式需和数据库中的表格式一致。

示例

将流audi_cheaper_than_30w的数据输出到数据库test的audi_cheaper_than_30w表下。

```
CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
    type = "rds",
    username = "root",
    password = "xxxxxx",
    db_url = "postgresql://192.168.1.1:8000/test",
    table_name = "audi_cheaper_than_30w"
  );

insert into audi_cheaper_than_30w select "1","2","3",4;
```

4.1.4.8 DWS 输出流（通过 OBS 转储方式）

功能描述

创建sink流将Flink作业数据通过OBS转储方式输出到数据仓库服务(DWS)，即Flink作业数据先输出到OBS，然后再从OBS导入到DWS。如何导入OBS数据到DWS具体可参考中“从OBS并行导入数据到集群”章节。

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。

注意事项

- 通过OBS转储支持两种中间文件方式：
  - ORC： ORC格式不支持Array数据类型，如果使用ORC格式，需先在DWS中创建外部服务器，具体可参考中“创建外部服务器”章节。
  - CSV： CSV格式默认记录分隔符为换行符，若属性内容中有换行符，建议配置quote，具体参见表4-12。
- 如果要写入的表不存在，则会自动创建表。由于DLI SQL类型不支持text，如果存在长文本，建议先在数据库中创建表。
- encode使用orc格式时，创建DWS表时，如果SQL流字段属性定义为String类型，DWS表字段属性不能使用varchar类型，需使用特定的text类型；如果是SQL流字段属性定义为Integer类型，DWS表字段需要使用Integer类型。

前提条件

- 确保已创建OBS桶和文件夹。
- 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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
        type = "dws",
        region = "",
        ak = "",
        sk = "",
        encode = "",
        field_delimiter = "",
        quote = "",
        db_obs_server = "",
        obs_dir = "",
        username = "",
        password =  "",
        db_url = "",
        table_name = "",
        max_record_num_per_file = "",
        dump_interval = ""
  );
```

关键字

表 4-12 关键字说明

参数	是否必选	说明
type	是	输出通道类型，dws表示输出到数据仓库服务中。
region	是	数据仓库服务所在区域。
ak	是	访问密钥ID(Access Key ID)。
sk	是	Secret Access Key，与访问密钥ID结合使用的密钥。
encode	是	编码方式。当前支持csv和orc两种方式。

参数	是否必选	说明
field_delimiter	否	属性分隔符。当编码方式为csv时需要配置，建议尽量用不可见字符作为分隔符，如\u0006\u0002。
quote	否	单字节，建议使用不可见字符，如\u0007。
db_obs_server	否	已在数据库中创建的外部服务器，如obs_server。 如果编码方式为orc格式时需指定该参数。
obs_dir	是	中间文件存储目录。格式为{桶名}/{目录名}， 如obs-a1/dir1/subdir。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址。格式为/ip:port/database，如“192.168.1.21:8000/test1”。
table_name	是	数据表名，若表不存在，则自动创建。
max_record_num_per_file	是	每个文件最多存储多少条记录。当文件记录数少于最大值时，该文件会延迟一个转储周期输出。
dump_interval	是	转储周期，单位为秒。
delete_obs_temp_file	否	是否要删除obs上的临时文件，默认为“true”，若设置为“false”，则不会删除obs上的文件，需用户自己清理。
max_dump_file_num	否	执行一次转储操作时最多转储多少文件。 当本次转储操作发现文件数小于最大值，则会延迟一个转储周期输出。

示例

- CSV格式转储。

```
CREATE SINK STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
  WITH (
    type = "dws",
    region = "xxx",
    ak = "",
    sk = "",
    encode = "csv",
    field_delimiter = "\u0006\u0006\u0002",
    quote = "\u0007",
    obs_dir = "dli-append-2/dws",
    username = "",
    password =  "",
    db_url = "192.168.1.12:8000/test1",
    table_name = "table1",
    max_record_num_per_file = "100",
    dump_interval = "10"
  );
```

- ORC格式转储。

```
CREATE SINK STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
  WITH (
    type = "dws",
    region = "xxx",
    ak = "",
    sk = "",
    encode = "orc",
    db_obs_server = "obs_server",
    obs_dir = "dli-append-2/dws",
    username = "",
    password =  "",
    db_url = "192.168.1.12:8000/test1",
    table_name = "table1",
    max_record_num_per_file = "100",
    dump_interval = "10"
  );
```

4.1.4.9 MRS HBase 输出流

功能描述

DLI将Flink作业的输出数据输出到MRS的HBase中。

前提条件

- 确保您的账户下已在MapReduce服务（MRS）里创建了您配置的集群。DLI支持与开启kerberos的hbase集群对接。
- 该场景作业需要运行在DLI的独享队列上，请确保已创建DLI独享队列。
- 确保DLI独享队列与MRS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。
- **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。**

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "mrs_hbase",
    region = "",
    cluster_address = "",
    table_name = "",
    table_columns = "",
    illegal_data_table = "",
    batch_insert_data_num = "",
    action = ""
)
```


关键字

表 4-13 关键字说明

参数	是否必选	说明
type	是	输出通道类型，"mrs_hbase"表示输出到MRS的HBase中。
region	是	MRS服务所在区域。
cluster_address	是	待插入数据表所属集群zookeeper地址，形如：ip1,ip2:port。
table_name	是	待插入数据的表名。 支持参数化，例如当需要某一列或者几列作为表名的一部分时，可表示为”car_pass_inspect_with_age_${car_age}“，其中car_age为列名。
table_columns	是	待插入的列，具体形式如："rowKey,f1:c1,f1:c2,f2:c1"，其中必须指定rowKey，当某列不需要加入数据库时，以第三列为例，可表示为"rowKey,f1:c1,,f2:c1"。
illegal_data_table	否	如果指定该参数，异常数据（比如：rowKey不存在）会写入该表（rowKey为taskNo加下划线加时间戳加六位随机数字，schema为info:data, info:reason），否则会丢弃。
batch_insert_data_num	否	表示一次性批量写入的数据条数，值必须为正整数，上限为1000，默认值为10。
action	否	表示数据是插入还是删除，可选值为add和delete，默认值为add。
krb_auth	否	创建跨源认证的认证名。开启kerberos认证时，需配置该参数，填写对应的跨源认证名称。 **说明** 请确保在DLI队列host文件中添加MRS集群master节点的“/etc/hosts”信息。

注意事项

无。

示例

将数据输出到MRS的HBase中。

```
CREATE SINK STREAM qualified_cars (
  car_id STRING,
  car_owner STRING,
  car_age INT,
  average_speed INT,
  total_miles INT
)
```

```
WITH (
  type = "mrs_hbase",
  region = "xxx",
  cluster_address = "192.16.0.88,192.87.3.88:2181",
  table_name = "car_pass_inspect_with_age_${car_age}",
  table_columns = "rowKey,info:owner,,car:speed,car:miles",
  illegal_data_table = "illegal_data",
  batch_insert_data_num = "20",
  action = "add",
  krb_auth = "KRB_AUTH_NAME"
);
```

4.1.4.10 MRS Kafka 输出流

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。MRS基于Apache Kafka在平台部署并托管了Kafka集群。

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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH(
    type = "kafka",
    kafka_bootstrap_servers = "",
    kafka_topic = "",
    encode = "json"
)
```

关键字

表 4-14 关键字说明

参数	是否必选	说明
type	是	输出通道类型，"kafka"表示输出到Kafka中。
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过增强型跨源开通DLI队列和Kafka集群的连接）。
kafka_topic	是	写入的topic。
encode	是	编码格式，当前支持“json”和“user_defined”。 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。

参数	是否必选	说明
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现编码类的类名（包含完整包路径），该类需继承类DeserializationSchema。
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现编码类的入参，仅支持一个string类型的参数。
krb_auth	否	创建跨源认证的认证名。开启kerberos认证时，需配置该参数。如果创建的MRS集群未开启kerb认证的集群，请确保在DLI队列host文件中添加MRS集群master节点的“/etc/hosts”信息。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"
kafka_certificate_name	否	跨源认证信息名称。跨源认证信息类型为“Kafka_SSL”时，该参数有效。 **说明** • 指定该配置项时，服务仅加载该认证下指定的文件和密码，系统将自动设置到“kafka_properties”属性中。 • Kafka SSL认证需要的其他配置信息，需要用户手动在“kafka_properties”属性中配置。

注意事项

无。

示例

将数据输出到Kafka中。

- 示例一

```
CREATE SINK STREAM kafka_sink (name STRING)
  WITH (
    type="kafka",
    kafka_bootstrap_servers =  "ip1:port1,ip2:port2",
    kafka_topic = "testsink",
    encode = "json"
  );
```

- 示例二

```
CREATE SINK STREAM kafka_sink (
  a1 string,
  a2 string,
  a3 string,
  a4 INT
) // 输出字段
  WITH (
    type="kafka",
    kafka_bootstrap_servers =  "192.x.x.x:9093, 192.x.x.x:9093, 192.x.x.x:9093",
    kafka_topic = "testflink", // 写入的topic
    encode = "csv", // 编码格式，支持json/csv
    kafka_certificate_name = "Flink",
    kafka_properties_delimiter = ",",
    kafka_properties = "sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
```



```
required username=\"xxx\" password=\"xxx\";,sasl.mechanism=PLAIN,security.protocol=SASL_SSL"
 );
```

4.1.4.11 开源 Kafka 输出流

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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
 WITH(
   type = "kafka",
   kafka_bootstrap_servers = "",
   kafka_topic = "",
   encode = "json"
 )
```

关键字

表 4-15 关键字说明

参数	是否必选	说明
type	是	输出通道类型，"kafka"表示输出到Kafka中。
kafka_bootstrap_servers	是	Kafka的连接端口，需要确保能连通（需要通过增强型跨源开通DLI队列和Kafka集群的连接）。
kafka_topic	是	写入的topic
encode	是	数据编码格式，可选为“csv”、“json”和“user_defined”。 ● 若编码格式为“csv”，则需配置“field_delimiter”属性。 ● 若编码格式为“user_defined”，则需配置“encode_class_name”和“encode_class_parameter”属性。
filed_delimiter	否	当encode为csv时，用于指定各字段分隔符，默认为逗号。
encode_class_name	否	当encode为user_defined时，需配置该参数，指定用户自实现编码类的类名（包含完整包路径），该类需继承类DeserializationSchema。

参数	是否必选	说明
encode_class_parameter	否	当encode为user_defined时，可以通过配置该参数指定用户自实现编码类的入参，仅支持一个string类型的参数。
kafka_properties	否	可通过该参数配置kafka的原生属性，格式为"key1=value1;key2=value2"
kafka_certificate_name	否	跨源认证信息名称。跨源认证信息类型为“Kafka_SSL”时，该参数有效。 **说明** • 指定该配置项时，服务仅加载该认证下指定的文件和密码，系统将自动设置到“kafka_properties”属性中。 • Kafka SSL认证需要的其他配置信息，需要用户手动在“kafka_properties”属性中配置。

注意事项

无。

示例

将流kafka_sink的数据输出到Kafka中。

```
CREATE SINK STREAM kafka_sink (name STRING)
  WITH (
    type="kafka",
    kafka_bootstrap_servers =  "ip1:port1,ip2:port2",
    kafka_topic = "testsink",
    encode = "json"
  );
```

4.1.4.12 文件系统输出流(推荐）

功能描述

创建sink流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。

对象存储服务（Object Storage Service，简称OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  [PARTITIONED BY (attr_name (',' attr_name)*]
  WITH (
    type = "filesystem",
    file.path = "obs://bucket/xx",
    encode = "parquet",
    ak = "",
    sk = ""
  );
```

关键字

表 4-16 关键字说明

参数	是否必选	说明
type	是	输出流类型。“type”为“filesystem”，表示输出数据到文件系统。
file.path	是	输出目录，格式为: schema://file.path。 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。 • 当schema为hdfs时，表示输出到HDFS。HDFS需要配置代理用户，具体请参考**HDFS代理用户配置**。 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。
encode	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。
ak	否	输出到OBS时该参数必填。用于访问OBS认证的accessKey，可使用全局变量，屏蔽敏感信息。
sk	否	输出到OBS时该参数必填。用于访问OBS认证的secretKey，可使用全局变量，屏蔽敏感信息。
krb_auth	否	创建跨源认证的认证名。开启kerberos认证时，需配置该参数。如果创建的MRS集群未开启kerb认证的集群，请确保在DLI队列host文件中添加MRS集群master节点的“/etc/hosts”信息。
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，如：“,”。

注意事项

- 使用文件系统输出流的Flink作业必须开启checkpoint，保证作业的一致性。
- 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或者手动重启，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。
- checkpoint支持如下两种模式：
  - AtLeastOnce：事件至少被处理一次。
  - ExactlyOnce：事件仅被处理一次。

- 使用文件系统输出流写入数据到OBS时，应避免多个作业写同一个目录的情况。
  - OBS对象存储桶的默认行为为覆盖写，可能导致数据丢失。
  - OBS并行文件系统桶的默认行为追加写，可能导致数据混淆。

  因为以上OBS桶类型行为的区别，为避免作业异常重启可能导致的数据异常问题，请根据您的业务需求选择OBS桶类型。

HDFS 代理用户配置

1. 登录MRS管理页面。
2. 选择MRS的HDFS Namenode配置，在“自定义”中添加配置参数。

   其中，core-site值名称“hadoop.proxyuser.myname.hosts”和“hadoop.proxyuser.myname.groups”中的“myname”为传入的krb认证用户名称。

   说明

   需要保证写入HDFS数据路径权限为777。
3. 配置完成后，单击“保存配置”进行保存。

示例

- 示例一：

  该示例将car_info数据，以buyday字段为分区字段，parquet为编码格式，转储数据到OBS。

```
create sink stream car_infos (
  carId string,
  carOwner string,
  average_speed double,
  buyday string
  ) partitioned by (buyday)
  with (
    type = "filesystem",
    file.path = "obs://obs-sink/car_infos",
    encode = "parquet",
    ak = "{{myAk}}",
    sk = "{{mySk}}"
);
```

  数据最终在OBS中的存储目录结构为：obs://obs-sink/car_infos/buyday=xx/part-x-x。

  数据生成后，可通过如下SQL语句建立OBS分区表，用于后续批处理：

  a. 创建OBS分区表。

```
create table car_infos (
  carId string,
  carOwner string,
  average_speed double
)
  partitioned by (buyday string)
  stored as parquet
  location 'obs://obs-sink/car_infos';
```

  b. 从关联OBS路径中恢复分区信息。

```
alter table car_infos recover partitions;
```

- 示例二

  该示例将car_info数据，以buyday字段为分区字段，csv为编码格式，转储数据到HDFS。

```
create sink stream car_infos (
  carId string,
```



```
  carOwner string,
  average_speed double,
  buyday string
  ) partitioned by (buyday)
  with (
    type = "filesystem",
    file.path = "hdfs://node-master1sYAx:9820/user/car_infos",
    encode = "csv",
    field_delimiter = ","
);
```

数据最终在HDFS中的存储目录结构为：/user/car_infos/buyday=xx/part-x-x。

4.1.4.13 OBS 输出流

功能描述

创建sink流将DLI数据输出到对象存储服务（OBS）。DLI可以将作业分析结果输出到OBS上。适用于大数据分析、原生云应用程序数据、静态网站托管、备份/活跃归档、深度/冷归档等场景。

对象存储服务（Object Storage Service，简称OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。OBS的更多信息，请参见。

说明

推荐使用《**文件系统输出流（推荐）**》。

前提条件

OBS输出流功能仅支持输出数据到3.0版本以上的桶，请先查看桶信息确认桶的版本。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
        type = "obs",
        region = "",
        encode = "",
        field_delimiter = "",
        row_delimiter = "",
        obs_dir = "",
        file_prefix = "",
        rolling_size = "",
        rolling_interval = "",
        quote = "",
        array_bracket = "",
        append = "",
        max_record_num_per_file = "",
        dump_interval = "",
        dis_notice_channel = ""
  )
```

关键字

表 4-17 关键字说明

参数	是否必选	说明
type	是	输出通道类型，“obs”表示输出到对象存储服务。
region	是	对象存储服务所在区域。
ak	否	访问密钥ID(Access Key ID)。
sk	否	Secret Access Key，与访问密钥ID结合使用的密钥。
encode	是	编码方式。当前支持csv/json/orc/avro/avro_merge/parquet格式。
field_delimiter	否	属性分隔符。 仅当编码方式为csv时需要配置，若不配置，默认分隔符为逗号。
row_delimiter	否	行分隔符。当编码格式为csv、json时需要设置。
json_config	否	当编码格式为json时，用户可以通过该参数来指定json字段和流定义字段的映射关系，格式为“field1=data_json.field1;field2=data_json.field2”。
obs_dir	是	文件存储目录。格式为{桶名}/{目录名}， 如obs-a1/dir1/subdir。当编码格式为csv（append为false）、json（append为false）、avro_merge、parquet时，支持参数化。
file_prefix	否	输出文件名前缀。生成的文件会以file_prefix.*x的方式命名，*如file_prefix.1、 file_prefix.2，若没有设置，默认文件前缀为temp。
rolling_size	否	单个文件最大允许大小。 **说明** • rolling_size和rolling_interval必须至少配一样或者都配置。 • 当文件大小超过设置size后，会生成新文件。 • 支持的单位包括KB/MB/GB，若没写单位，表示单位为字节数。 • 当编码格式为orc时不需要设置。

参数	是否必选	说明
rolling_interval	否	数据保存到对应目录的时间模式。 **说明** • rolling_size和rolling_interval必须至少配一样或者都配置。 • 设置后数据会按照输出时间输出到相应时间目录下。 • 支持的格式为yyyy/MM/dd/HH/mm， 最小单位只到分钟，大小写敏感。例如配置为yyyy/MM/dd/HH， 则数据会写入对应小时这个时间点所产生的目录下，比如2018-09-10 16时产生的数据就会写到{obs_dir}/2018-09-10_16目录下。 • 当rolling_size和rolling_interval都配置时，表示每个时间所对应的目录下，单个文件超过设置大小时，另起新文件。
quote	否	修饰符，仅当编码格式为csv时可配置，配置后会在每个属性前后各加上修饰符，建议使用不可见字符配置，如"\u0007"。
array_bracket	否	数组括号，仅当编码格式为csv时可配置， 可选值为"()", "{}", "[]"， 例如配置了"{}", 则数组输出格式为{a1,a2}。
append	否	值为true或者false，默认为true。 当OBS不支持append模式，且编码格式为csv和json时，可将该参数设置为false。Append为false时需要设置max_record_num_per_file和dump_interval。
max_record_num_per_file	否	文件最大记录数，当编码格式为csv（append为false）、json（append为false）、orc、avro、avro_merge和parquet时需配置，表明一个文件最多存储记录数，当达到最大值，则另起新文件。
dump_interval	否	触发周期， 当编码格式为orc或者配置了DIS通知提醒时需进行配置。 • 在orc编码方式中，该配置表示周期到达时，即使文件记录数未达到最大个数配置，也将文件上传到OBS上。 • 在DIS通知提醒功能中，该配置表示每周期往DIS发送一个通知提醒，表明该目录已写完。
dis_notice_channel	否	OBS目录完成通知通道。表示每周期往DIS通道中发送一条记录，该记录内容为OBS目录路径，表明该目录已书写完毕。
encoded_data	否	当编码格式为json（append为false）、avro_merge和parquet时，可通过配置该参数指定真正需要编码的数据，格式为${field_name}，表示直接将该流字段的内容作为一个完整的记录进行编码。

注意事项

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。

示例

- 将car_infos数据输出到OBS的obs-sink桶下，输出目录为car_infos， 输出文件以greater_30作为文件名前缀，当单个文件超过100M时新起一个文件，同时数据输出用csv编码，使用逗号作为属性分隔符，换行符作为行分隔符。

```
CREATE SINK STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
  WITH (
    type = "obs",
    encode = "csv",
    region = "xxx",
    field_delimiter = ",",
    row_delimiter = "\n",
    obs_dir = "obs-sink/car_infos",
    file_prefix = "greater_30",
    rolling_size = "100m"
);
```

- orc编码格式示例

```
CREATE SINK STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT,
  car_timestamp LONG
)
  WITH (
    type = "obs",
    region = "xxx",
    encode = "orc",
    obs_dir = "dli-append-2/obsorc",
    FILE_PREFIX = "es_info",
    max_record_num_per_file = "100000",
    dump_interval = "60"
);
```

- parquet编码示例请参考**文件系统输出流(推荐）**中的示例。

4.1.4.14 RDS 输出流

功能描述

DLI将Flink作业的输出数据输出到关系型数据库（RDS）中。目前支持PostgreSQL和MySQL两种数据库。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。MySQL数据库适用于各种WEB应用、电子商务应用、企业应用、移动应用等场景，减少IT部署和维护成本。

关系型数据库（Relational Database Service，简称RDS）是一种基于云计算平台的在线关系型数据库服务。

前提条件

- 请务必确保您的账户下已在关系型数据库（RDS）里创建了PostgreSQL或MySQL类型的RDS实例。
- 该场景作业需要运行在DLI的独享队列上，因此要与RDS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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "rds",
    username = "",
    password = "",
    db_url = "",
    table_name = ""
  );
```

关键字

表 4-18 关键字说明

参数	是否必选	说明
type	是	输出通道类型，rds表示输出到关系型数据库中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址，格式为："{database_type}://ip:port/database" 目前支持两种数据库连接：MySQL和PostgreSQL ● MySQL: 'mysql://ip:port/database' ● PostgreSQL: 'postgresql://ip:port/database'
table_name	是	要插入数据的数据库表名。

db_columns	否	支持配置输出流属性和数据库表属性的对应关系，需严格按照输出流的属性顺序配置。
		示例：
		`create sink stream a3(student_name  string, student_age int) with (`
		`type = "rds",`
		`username = "root",`
		`password = "xxxxxxxx",`
		`db_url = "mysql://192.168.0.102:8635/test1",`
		`db_columns = "name,age",`
		`table_name = "t1"`
		`);`
		student_name对应数据库里的name属性，student_age对应数据库里的age属性。
		说明
		● 当不配置db_columns时，若输出流属性个数小于数据库表属性个数，并且数据库多出的属性都是nullable或者有默认值时，这种情况也允许。

参数	是否必选	说明
primary_key	否	如果想通过主键实时更新表中的数据，需要在创建数据表的时候增加primary_key配置项，如下面例子中的c_timeminute。配置primary_key后，在进行数据写入操作时，如果primary_key存在，则进行更新操作，否则进行插入操作。 示例： CREATE SINK STREAM test(c_timeminute LONG, c_cnt LONG) WITH (type = "rds", username = "root", password = "xxxxxxxx", db_url = "mysql://192.168.0.12:8635/test", table_name = "test", primary_key = "c_timeminute");
operation_field	否	该配置项用于指定数据的处理方式，需要配置为${field_name}的形式，field_name的类型必读为string，field_name所代表的真正内容表示为D或者DELETE时，表示删除数据库中该条记录，其余默认插入数据。

注意事项

stream_id所定义的流格式需和数据库中的表格式一致。

示例

将流audi_cheaper_than_30w的数据输出到数据库test的audi_cheaper_than_30w表下。

```
CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
    type = "rds",
    username = "root",
    password = "xxxxxx",
    db_url = "mysql://192.168.1.1:8635/test",
    table_name = "audi_cheaper_than_30w"
);
```

4.1.4.15 SMN 输出流

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。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH(
    type = "smn",
    region = "",
    topic_urn = "",
    urn_column = "",
    message_subject = "",
    message_column = ""
  )
```

关键字

表 4-19 关键字说明

参数	是否必选	说明
type	是	输出通道类型，smn表示输出到消息通知服务中。
region	是	SMN所在区域。
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。
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。
message_subject	是	发往SMN服务的消息标题，用户自定义。
message_column	是	输出流的字段名，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

将流over_speed_warning的数据输出到消息通知服务SMN中。

```
//静态主题配置
CREATE SINK STREAM over_speed_warning (
  over_speed_message STRING /* over speed message */
)
  WITH (
    type = "smn",
    region = "xxx",
    topic_Urn = "xxx",
    message_subject = "message title",
    message_column = "over_speed_message"
  );
```

```
//动态主题配置
CREATE SINK STREAM over_speed_warning2 (
    over_speed_message STRING, /* over speed message */
    over_speed_urn STRING
)
  WITH (
    type = "smn",
    region = "xxx",
    urn_column = "over_speed_urn",
    message_subject = "message title",
    message_column = "over_speed_message"
  );
```

4.1.5 创建中间流

功能描述

中间流用来简化sql逻辑，若sql逻辑比较复杂，可以写多个sql语句，用中间流进行串接。中间流仅为逻辑意义上的流，不会产生数据存储。

语法格式

创建中间流语法格式如下：

```
CREATE TEMP STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
```

示例

创建中间流示例如下：

```
create temp stream a2(attr1 int, attr2 string);
```

4.1.6 创建维表

4.1.6.1 创建 Redis 表

创建Redis表用于与输入流连接。

流表JOIN语法请参见**流表JOIN**。

语法格式

```
CREATE TABLE table_id (key_attr_name STRING(, hash_key_attr_name STRING)?, value_attr_name STRING)
  WITH (
    type = "dcs_redis",
    cluster_address = ""(,password = "")?,
    value_type= "",
    key_column= ""(,hash_key_column="")?);
```

关键字

表 4-20 关键字说明

参数	是否必选	说明
type	是	输出通道类型，dcs_redis表示输出到分布式缓存服务的Redis存储系统中。
cluster_address	是	Redis实例连接地址。
password	否	Redis实例连接密码，当设置为免密访问时，省略该配置项。
value_type	是	指定数据类型。支持的数据类型包括：string, list, hash, set, zset。
key_column	是	指定代表Redis key属性的列名。
hash_key_column	否	当value_type设置为hash时，需要指定本字段作为第二级key属性的列名。
cache_max_num	否	表示最大缓存的查询结果数，默认值为32768。
cache_time	否	表示数据库查询结果在内存中缓存的最大时间。单位为毫秒，默认值为10000，当值为0时表示不缓存。

注意事项

- 不支持Redis集群。
- 请务必确保您的账户下已在分布式缓存服务（DCS）里创建了Redis类型的缓存实例。
- 该场景作业需要运行在DLI的独享队列上，因此要与DCS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。

示例

Redis表用于与输入流连接。

```
CREATE TABLE table_a (attr1 string, attr2 string, attr3 string)
  WITH (
    type = "dcs_redis",
    value_type = "hash",
    key_column = "attr1",
    hash_key_column = "attr2",
    cluster_address = "192.168.1.238:6379",
    password = "xxxxxxxx"
 );
```

4.1.6.2 创建 RDS 表

创建RDS/DWS表用于与输入流连接。

流表JOIN语法请参见**流表JOIN**。

前提条件

- 请务必确保您的账户下已在关系型数据库（RDS）里创建了PostgreSQL或MySQL类型的RDS实例。
- 该场景作业需要运行在DLI的独享队列上，因此要与RDS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。

语法格式

```
CREATE TABLE  table_id (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
    type = "rds",
    username = "",
    password = "",
    db_url = "",
    table_name = ""
  );
```

关键字

表 4-21 关键字说明

参数	是否必选	说明
type	是	输出通道类型，rds表示输出到关系型数据库中。
username	是	数据库连接用户名。
password	是	数据库连接密码。
db_url	是	数据库连接地址，格式为："{database_type}://ip:port/database" 目前支持两种数据库连接：MySQL和PostgreSQL • MySQL: 'mysql://ip:port/database' • PostgreSQL: 'postgresql://ip:port/database' **说明** 将数据库连接地址设置为DWS数据库地址，即可创建DWS维表。DWS数据库版本大于8.1.0后，无法用开源的postgresql驱动连接，需要用gaussdb驱动进行连接。
table_name	是	用于查询数据的数据库表名。
db_columns	否	流属性和数据库表的字段对应关系。当sink流中流属性字段和数据库表中的流属性字段不完全匹配时，该参数必配。格式为“dbtable_attr1,dbtable_attr2,dbtable_attr3“。

参数	是否必选	说明
cache_max_num	否	表示最大缓存的查询结果数，默认值为32768。
cache_time	否	表示数据库查询结果在内存中缓存的最大时间。单位为毫秒，默认值为10000，当值为0时表示不缓存。

示例

RDS表用于与输入流连接。

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
    type = "dis",
    region = "",
    channel = "dliinput",
    encode = "csv",
    field_delimiter = ","
  );

CREATE TABLE  db_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
    type = "rds",
    username = "root",
    password = "******",
    db_url = "postgresql://192.168.0.0:2000/test1",
    table_name = "car"
);

CREATE SINK STREAM audi_cheaper_than_30w (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_price INT
)
  WITH (
    type = "dis",
    region = "",
    channel = "dlioutput",
    partition_key = "car_owner",
    encode = "csv",
    field_delimiter = ","
  );

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 as b on a.car_id = b.car_id;
```

说明

将数据库连接地址设置为DWS数据库地址，即可创建DWS维表。DWS数据库版本大于8.1.0后，无法用开源的postgresql驱动连接，需要用gaussdb驱动进行连接。

4.1.7 自拓展生态

4.1.7.1 自拓展输入流

用户可通过编写代码实现从想要的云生态或者开源生态获取数据，作为Flink作业的输入数据。

语法格式

```
CREATE SOURCE STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
  WITH (
    type = "user_defined",
    type_class_name = "",
    type_class_parameter = ""
  )
  (TIMESTAMP BY timeindicator (',' timeindicator)?);timeindicator:PROCTIME '.' PROCTIME| ID '.' ROWTIME
```

关键字

表 4-22 关键字说明

参数	是否必选	说明
type	是	数据源类型，"user_defined"表示数据源为用户自定义数据源。
type_class_name	是	用户实现获取源数据的source类名称，注意包含完整包路径。
type_class_parameter	是	用户自定义source类的入参，仅支持一个string类型的参数。

注意事项

用户自定义source类需要继承类RichParallelSourceFunction，并指定数据类型为Row
例如定义类MySource：public class MySource extends RichParallelSourceFunction<Row>{}，重点实现其中的open、run和close函数。

依赖pom:

```
<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>
```

示例

实现每周期产生一条数据（仅包含一个字段，类型为INT，初始值为1，每周期加1），周期时长为60s，通过入参指定。

```
CREATE SOURCE STREAM user_in_data (
   count INT
   )
 WITH (
  type = "user_defined",
  type_class_name = "mySourceSink.MySource",
  type_class_parameter = "60"
   )
   TIMESTAMP BY car_timestamp.rowtime;
```

说明

自定义source类实现，需要将该类打在jar包中，通过sql编辑页上传udf函数按钮上传。

4.1.7.2 自拓展输出流

用户可通过编写代码实现将DLI处理之后的数据写入指定的云生态或者开源生态。

语法格式

```
CREATE SINK STREAM stream_id (attr_name attr_type (',' attr_name attr_type)* )
 WITH (
  type = "user_defined",
  type_class_name = "",
  type_class_parameter = ""
);
```

关键字

表 4-23 关键字说明

参数	是否必选	说明
type	是	数据源类型，"user_defined"表示数据源为用户自定义数据源。
type_class_name	是	用户实现获取源数据的sink类名称，注意包含完整包路径。
type_class_parameter	是	用户自定义sink类的入参，仅支持一个string类型的参数。

注意事项

用户自定义sink类需要继承类RichSinkFunction，并指定数据类型为Row例如定义类MySink：public class MySink extends RichSinkFunction<Row>{}，重点实现其中的open、invoke和close函数。

依赖pom:

```
<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>
```

示例

实现数据以CSV编码写入DIS通道。

```
CREATE SINK STREAM user_out_data (
    count INT
)
  WITH (
    type = "user_defined",
    type_class_name = "mySourceSink.MySink",
    type_class_parameter = ""
      );
```

说明

自定义sink类实现，需要将该类打在jar包中，通过sql编辑页上传udf函数按钮上传。

4.1.8 数据类型

概述

数据类型是数据的一个基本属性，用于区分不同类别的数据。不同的数据类型所占的存储空间不同，能够进行的操作也不相同。数据库中的数据存储在数据表中。数据表中的每一列都定义了数据类型，用户存储数据时，须遵从这些数据类型的属性，否则可能会出错。

大数据平台的Flink SQL与开源社区相同，支持原生数据类型、复杂数据类型和复杂类型嵌套。

原生数据类型

Flink SQL支持原生数据类型，请参见表4-24。

表 4-24 原生数据类型

数据类型	描述	存储空间	范围
VARCHAR	可变长度的字符	-	-
BOOLEAN	布尔类型	-	TRUE/FALSE
TINYINT	有符号整数	1字节	-128-127
SMALLINT	有符号整数	2字节	-32768-32767
INT	有符号整数	4字节	-2147483648～2147483647

数据类型	描述	存储空间	范围
INTEGER	有符号整数	4字节	-2147483648～2147483647
BIGINT	有符号整数	8字节	-9223372036854775808～9223372036854775807
REAL	单精度浮点型	4字节	-
FLOAT	单精度浮点型	4字节	-
DOUBLE	双精度浮点型	8字节	-
DECIMAL	固定有效位数和小数位数的数据类型	-	-
DATE	日期类型，描述了特定的年月日，以yyyy-MM-dd格式表示，例如2014-05-29	-	DATE类型不包含时间，所表示日期的范围为0000-01-01 to 9999-12-31
TIME	时间类型，以HH:mm:ss表示。 例如20:17:40	-	-
TIMESTAMP(3)	完整日期，包括日期和时间。 例如：1969-07-20 20:17:40	-	-
INTERVAL timeUnit [TO timeUnit]	时间间隔 例如：INTERVAL '1:5' YEAR TO MONTH, INTERVAL '45' DAY	-	-

复杂数据类型

Flink SQL支持复杂数据类型和复杂类型嵌套。复杂数据类型如**表4-25**所示。

表 4-25 复杂数据类型

数据类型	描述	声明方式	引用方式	构造方式
ARRAY	一组有序字段，所有字段的数据类型必须相同。	ARRAY[TYPE]	变量名[下标]，下标从1开始，例如：v1[1]	Array[value1, value2, ...] as v1

数据类型	描述	声明方式	引用方式	构造方式
MAP	一组无序的键/值对。键的类型必须是原生数据类型，值的类型可以是原生数据类型或复杂数据类型。同一个MAP键的类型必须相同，值的类型也必须相同。	MAP[TYPE，TYPE]	变量名[key]，例如：v1[key]	Map[key, value, key2, value2, key3, value3.....] as v1
ROW	一组命名的字段，字段的数据类型可以不同。	ROW<a1 TYPE1, a2 TYPE2>	变量名.字段名，例如：v1.a1	Row('1',2) as v1

使用示例如下：

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  address ROW<city STRING, province STRING, country STRING>,
  average_speed MAP[STRING, LONG],
  speeds ARRAY[LONG]
)
  WITH (
    type = "dis",
    region = "xxx",
    channel = "dliinput",
    encode = "json"
);

CREATE temp STREAM car_speed_infos (
  car_id STRING,
  province STRING,
  average_speed LONG,
  start_speed LONG
);

INSERT INTO car_speed_infos SELECT
   car_id,
   address.province,
   average_speed[address.city],
   speeds[1]
FROM car_infos;
```

复杂类型嵌套

- Json格式增强

  以Source为例进行说明，Sink的使用方法相同。

  - 支持配置Json_schema

    配置了json_schema后，可以不声明DDL中的字段，自动从json_schema中生成。使用示例如下：

    ```
    CREATE SOURCE STREAM data_with_schema WITH (
          type = "dis",
          region = "xxx",
          channel = "dis-in",
          encode = "json",
          json_schema = '{"definitions":{"address":{"type":"object","properties":{"street_address":
    {"type":"string"},"city":{"type":"string"},"state":{"type":"string"}},"required":
    ["street_address","city","state"]}},"type":"object","properties":{"billing_address":{"$ref":"#/
    ```



```
definitions/address"},"shipping_address":{"$ref":"#/definitions/address"},"optional_address":
{"oneOf":[{"type":"null"},{"$ref":"#/definitions/address"}]}}}'
    );

    CREATE SINK STREAM buy_infos (
      billing_address_city STRING,
      shipping_address_state string
    ) WITH (
      type = "obs",
      encode = "csv",
      region = "xxx" ,
      field_delimiter = ",",
      row_delimiter = "\n",
      obs_dir = "bucket/car_infos",
      file_prefix = "over",
      rolling_size = "100m"
    );

    insert into buy_infos select billing_address.city, shipping_address.state from
data_with_schema;
```

示例数据：

```
{
 "billing_address":
  {
   "street_address":"xxx",
   "city":"xxx",
   "state":"xxx"
   },
 "shipping_address":
  {
   "street_address":"xxx",
   "city":"xxx",
   "state":"xxx"
  }
}
```

- 支持不配置json_schema也不配置json_config。json_config使用可以参考**开源Kafka输入流**样例说明。

  这种情况下默认用ddl中属性名当做json key来进行解析。

  测试示例数据如下，测试数据既包括嵌套json字段，如billing_address、shipping_address，也包括非嵌套的字段id、type2。

  ```
  {
   "id":"1",
   "type2":"online",
   "billing_address":
   {
    "street_address":"xxx",
    "city":"xxx",
    "state":"xxx"
    },
   "shipping_address":
   {
    "street_address":"xxx",
    "city":"xxx",
    "state":"xxx"
   }
  }
  ```

  具体建表和使用示例参考如下：

  ```
  CREATE SOURCE STREAM car_info_data (
      id STRING,
      type2 STRING,
      billing_address Row<street_address string, city string, state string>,
      shipping_address Row<street_address string, city string, state string>,
      optional_address Row<street_address string, city string, state string>
    ) WITH (
      type = "dis",
  ```



```
    region = "xxx",
    channel = "dis-in",
    encode = "json"
  );

  CREATE SINK STREAM buy_infos (
    id STRING,
    type2 STRING,
    billing_address_city STRING,
    shipping_address_state string
  ) WITH (
    type = "obs",
    encode = "csv",
    region = "xxx",
    field_delimiter = ",",
    row_delimiter = "\n",
    obs_dir = "bucket/car_infos",
    file_prefix = "over",
    rolling_size = "100m"
  );

    insert into buy_infos select id, type2, billing_address.city, shipping_address.state from
car_info_data;
```

- Sink序列化支持复杂类型
  - 目前只有CSV、Json两种格式支持复杂类型。
  - Json请参考**•Json格式增强**。
  - 由于CSV没有标准的格式，所以目前暂不支持source解析，只支持sink。
  - 输出格式：尽量和flink原生保持一致。

    Map: {key1=Value1, key2=Value2}

    Row: 平摊用逗号分隔属性，如Row(1, '2') => 1,'2'

4.1.9 内置函数

4.1.9.1 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4-26**。

表 4-26 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。

运算符	返回类型	描述
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)。
A NOT BETWEEN B AND C	BOOLEAN	若A小于B或大于C，则返回TRUE。
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。

说明

- 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4-27，表中的A和B代表逻辑表达式。

表 4-27 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。

说明

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4-28所示。

表 4-28 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。
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。

运算符	返回类型	描述
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。
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。

说明

字符串类型不能参与算术运算。

4.1.9.2 字符串函数

DLI常用字符串函数如下所示：

表 4-29 字符串运算符

运算符	返回类型	描述
||	VARCHAR	两个字符串的拼接。
CHAR_LENGTH	INT	返回字符串中的字符数量。

运算符	返回类型	描述
CHARACTER_LENGTH	INT	返回字符串中的字符数量。
CONCAT	VARCHAR	拼接两个或多个字符串值从而组成一个新的字符串。如果任一参数为NULL时，则跳过该参数。
CONCAT_WS	VARCHAR	将每个参数值和第一个参数separator指定的分隔符依次连接到一起组成新的字符串，长度和类型取决于输入值。
HASH_CODE	INT	返回字符串的HASH_CODE()的绝对值。参数除string外，也支持int/bigint/float/double。
INITCAP	VARCHAR	返回字符串，将单词首字母转换为大写，其余为小写。 单词是由非字母、数字、字符分隔的字母、数字、字符序列。
IS_ALPHA	BOOLEAN	判断字符串是否只包含字母。
IS_DIGITS	BOOLEAN	判断字符串是否只包含数字。
IS_NUMBER	BOOLEAN	判断字符串是否是数值。
IS_URL	BOOLEAN	判断字符串是否是合法的URL地址。
JSON_VALUE	VARCHAR	获取json字符串中指定path的值。
KEY_VALUE	VARCHAR	获取键值对字符串中某一个key对应的值。
LOWER	VARCHAR	返回小写字符的字符串。
LPAD	VARCHAR	将pad字符串拼接到str字符串的的左端，直到新的字符串达到指定长度len为止。
MD5	VARCHAR	返回字符串的MD5值。如果参数为空串（即参数为''）时，则返回空串。
OVERLAY	VARCHAR	用y替换x的子串。从start_position开始，替换length+1个字符。
POSITION	INT	返回目标字符串x在被查询字符串y里第一次出现的位置。如果目标字符串x在被查询字符串y中不存在，返回值为0。

运算符	返回类型	描述
REPLACE	VARCHAR	字符串替换函数，将字符串str1中的所有str2替换成str3。 • str1：原字符。 • str2：目标字符。 • str3：替换字符。
RPAD	VARCHAR	将pad字符串拼接到str字符串的的右端，直到新的字符串达到指定长度len为止。
SHA1	STRING	返回字符串expr的SHA1值。
SHA256	STRING	返回字符串expr的SHA256值。
STRING_TO_ARRAY	ARRAY[STRING]	将字符串value按delimiter分隔为字符串数组。
SUBSTRING	VARCHAR	返回从给定位置开始的A的子字符串。起始位置从1开始。
TRIM	STRING	从B中除去字符串首尾/首位/末尾的A。默认情况下，首尾的A都被删除。
UPPER	VARCHAR	返回转换为大写字符的字符串。

||

- 功能描述

  两个字符串的拼接。
- 语法

  ```
  VARCHAR VARCHAR a || VARCHAR b
  ```
- 参数说明
  - a：字符串。
  - b：字符串。
- 示例
  - 测试语句

    ```
    SELECT "hello" || "world";
    ```
  - 测试结果

    ```
    "helloworld"
    ```

CHAR_LENGTH

- 功能描述

  返回字符串中的字符数量。
- 语法

  ```
  INT CHAR_LENGTH(a)
  ```
- 参数说明

  - a：字符串。
- 示例
  - 测试语句
    ```
    SELECT  CHAR_LENGTH(var1) as aa FROM T1;
    ```
  - 测试数据和结果

    表 4-30 测试数据和结果

测试数据（var1）	测试结果（aa）
abcde123	8

CHARACTER_LENGTH

- 功能描述

  返回字符串中的字符数量。
- 语法
  ```
  INT CHARACTER_LENGTH(a)
  ```
- 参数说明
  - a：字符串。
- 示例
  - 测试语句
    ```
    SELECT  CHARACTER_LENGTH(var1) as aa FROM T1;
    ```
  - 测试数据和结果

    表 4-31 测试数据和结果

测试数据（var1）	测试结果（aa）
abcde123	8

CONCAT

- 功能描述

  拼接两个或多个字符串值从而组成一个新的字符串。如果任一参数为NULL时，则跳过该参数。
- 语法
  ```
  VARCHAR CONCAT(VARCHAR var1, VARCHAR var2, ...)
  ```
- 参数说明
  - var1：字符串
  - var2：字符串
- 示例
  - 测试语句
    ```
    SELECT CONCAT("abc", "def", "ghi", "jkl");
    ```
  - 测试结果
    ```
    "abcdefghijkl"
    ```


CONCAT_WS

- 功能描述

  将每个参数值和第一个参数separator指定的分隔符依次连接到一起组成新的字符串，长度和类型取决于输入值。

  说明

  如果separator取值为null，则将separator视作与空串进行拼接。如果其它参数为null，在执行拼接过程中跳过取值为null的参数。

- 语法

  ```
  VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
  ```

- 参数说明
  - separator：分隔符。
  - var1：字符串。
  - var2：字符串。
- 示例
  - 测试语句

    ```
    SELECT CONCAT_WS("-", "abc", "def", "ghi", "jkl");
    ```

  - 测试结果

    ```
    "abc-def-ghi-jkl"
    ```

HASH_CODE

- 功能描述

  返回字符串的HASH_CODE()的绝对值。参数除string外，也支持int/bigint/float/double。

- 语法

  ```
  INT HASH_CODE(VARCHAR str)
  ```

- 参数说明
  - str：字符串。
- 示例
  - 测试语句

    ```
    SELECT HASH_CODE("abc");
    ```

  - 测试结果

    ```
    96354
    ```

INITCAP

- 功能描述

  返回字符串，将字符串首字母转换为大写，其余为小写。字符串是由非字母、数字、字符分隔的字母、数字、字符序列。

- 语法

  ```
  VARCHAR INITCAP(a)
  ```

- 参数说明
  - a：字符串。
- 示例
  - 测试语句

    ```
    SELECT INITCAP(var1)as aa FROM T1;
    ```


- 测试数据和结果

表 4-32 测试数据和结果

测试数据（var1）	测试结果（aa）
aBCde	Abcde

IS_ALPHA

- 功能描述

  判断字符串是否只包含字母。

- 语法

```
BOOLEAN IS_ALPHA(VARCHAR content)
```

- 参数说明
  - content：输入字符串。
- 示例
  - 测试语句

```
SELECT IS_ALPHA(content)  AS case_result FROM T1;
```

  - 测试数据和结果

表 4-33 测试数据和结果

测试数据（content）	测试结果（case_result）
Abc	true
abc1#$	false
null	false
""(空字符串)	false

IS_DIGITS

- 功能描述

  判断字符串是否只包含数字。

- 语法

```
BOOLEAN IS_DIGITS(VARCHAR content)
```

- 参数说明
  - content：输入字符串。
- 示例
  - 测试语句

```
SELECT IS_DIGITS(content) AS case_result FROM T1;
```

  - 测试数据和结果

表 4-34 测试数据和结果

测试数据（content）	测试结果（case_result）
78	true
78.0	false
78a	false
null	false
"" (空字符串)	false

IS_NUMBER

- 功能描述

  判断字符串是否是数值。

- 语法

  ```
  BOOLEAN IS_NUMBER(VARCHAR content)
  ```

- 参数说明
  - content：输入字符串。
- 示例
  - 测试语句

    ```
    SELECT IS_NUMBER(content) AS case_result FROM T1;
    ```

  - 测试数据和结果

    表 4-35 测试数据和结果

测试数据（content）	测试结果（case_result）
78	true
78.0	true
78a	false
null	false
"" (空字符串)	false

IS_URL

- 功能描述

  判断字符串是否是合法的URL地址。

- 语法

  ```
  BOOLEAN IS_URL(VARCHAR content)
  ```

- 参数说明
  - content：输入字符串。
- 示例

- 测试语句
  ```
  SELECT IS_URL(content) AS case_result FROM T1;
  ```
- 测试数据和结果

表 4-36 测试数据和结果

测试数据（content）	测试结果（case_result）
https://www.testweb.com	true
https://www.testweb.com:443	true
www.testweb.com:443	false
null	false
"" (空字符串)	false

JSON_VALUE

- 功能描述

  获取json字符串中指定path的值。
- 语法
  ```
  VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
  ```
- 参数说明
  - content：输入字符串。
  - path：要获取的path路径。
- 示例
  - 测试语句
    ```
    SELECT JSON_VALUE(content, path) AS case_result FROM T1;
    ```
  - 测试数据和结果

表 4-37 测试数据和结果

测试数据（content，path）		测试结果（case_result）
{ "a1":"v1","a2":7,"a3":8.0,"a4": {"a41":"v41","a42": ["v1","v2"]}}	$	{ "a1":"v1","a2":7,"a3":8.0,"a4": {"a41":"v41","a42": ["v1","v2"]}}
{ "a1":"v1","a2":7,"a3":8.0,"a4": {"a41":"v41","a42": ["v1","v2"]}}	$.a1	v1
{ "a1":"v1","a2":7,"a3":8.0,"a4": {"a41":"v41","a42": ["v1","v2"]}}	$.a4	{"a41":"v41","a42": ["v1","v2"]}
{ "a1":"v1","a2":7,"a3":8.0,"a4": {"a41":"v41","a42": ["v1","v2"]}}	$.a4.a42	["v1","v2"]

测试数据（content，path）		测试结果（case_result）
{ "a1":"v1","a2":7,"a3":8.0,"a4" : {"a41":"v41","a42": ["v1","v2"]}}	$.a4.a42[0]	v1

KEY_VALUE

- 功能描述

  获取键值对字符串中某一个key对应的值。

- 语法

  ```
  VARCHAR KEY_VALUE(VARCHAR content, VARCHAR split1, VARCHAR split2, VARCHAR key_name)
  ```

- 参数说明
  - content：输入字符串。
  - split1：多个键值对分隔符。
  - split2：key/value分隔符。
  - key_name：要获取的键名称。
- 示例
  - 测试语句

    ```
    SELECT KEY_VALUE(content, split1, split2, key_name)  AS case_result FROM T1;
    ```

  - 测试数据和结果

    表 4-38 测试数据和结果

测试数据（content, split1, split2, key_name）				测试结果（case_result）
k1=v1;k2=v2	;	=	k1	v1
null	;	=	k1	null
k1=v1;k2=v2	null	=	k1	null

LOWER

- 功能描述

  返回小写字符的字符串。

- 语法

  ```
  VARCHAR LOWER(A)
  ```

- 参数说明
  - A：字符串。
- 示例
  - 测试语句

    ```
    SELECT LOWER(var1) AS aa FROM T1;
    ```

  - 测试数据和结果

表 4-39 测试数据和结果

测试数据（var1）	测试结果（aa）
ABc	abc

LPAD

- 功能描述

  将pad字符串拼接到str字符串的的左端，直到新的字符串达到指定长度len为止。

- 语法

  ```
  VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
  ```

- 参数说明
  - str：拼接前的字符串。
  - len：拼接后的字符串的长度。
  - pad：被拼接的字符串。

  说明

  - 任意参数为null时返回null。
  - len为负数时返回为null。
  - len不大于str长度时，返回str裁剪为len长度的字符串。

- 示例
  - 测试语句

    ```
    SELECT
      LPAD("adc", 2, "hello"),
      LPAD("adc", -1, "hello"),
      LPAD("adc", 10, "hello");
    ```

  - 测试结果

    ```
    "ad",,"helloheadc"
    ```

MD5

- 功能描述

  返回字符串的MD5值。如果参数为空串（即参数为''）时，则返回空串。

- 语法

  ```
  VARCHAR MD5(VARCHAR str)
  ```

- 参数说明
  - str：字符串

- 示例
  - 测试语句

    ```
    SELECT MD5("abc");
    ```

  - 测试结果

    ```
    "900150983cd24fb0d6963f7d28e17f72"
    ```

OVERLAY

- 功能描述

  用y替换x的子串。从start_position开始，替换length+1个字符。

- 语法
  ```
  VARCHAR OVERLAY ( (VARCHAR x PLACING VARCHAR y FROM INT start_position [ FOR INT length ]) )
  ```
- 参数说明
  - x：字符串。
  - y：字符串。
  - start_position：起始位置。
  - length（可选）：字符长度。
- 示例
  - 测试语句：
    ```
    OVERLAY('abcdefg' PLACING 'xyz' FROM 2 FOR 2) AS result FROM T1;
    ```
  - 测试结果：

    表 4-40 测试结果

result
axyzdefg

POSITION

- 功能描述

  返回目标字符串x在被查询字符串y里第一次出现的位置。如果目标字符串x在被查询字符串y中不存在，返回值为0。
- 语法
  ```
  INTEGER POSITION(x IN y)
  ```
- 参数说明
  - x：字符串。
  - y：字符串。
- 示例
  - 测试语句：
    ```
    POSITION('in' IN 'chin') AS result FROM T1;
    ```
  - 测试结果

    表 4-41 测试结果

result
3

REPLACE

- 功能描述

  字符串替换函数，将字符串str1中的所有str2替换成str3。
- 语法
  ```
  VARCHAR REPLACE(VARCHAR str1, VARCHAR str2, VARCHAR str3)
  ```

- 参数说明
  - str1：原字符。
  - str2：目标字符。
  - str3：替换字符。
- 示例
  - 测试语句：
    ```
    SELECT
      replace(
        "hello world hello world hello world",
        "world",
        "hello"
      );
    ```
  - 测试结果
    ```
    "hello hello hello hello hello hello"
    ```

RPAD

- 功能描述

  将pad字符串拼接到str字符串的的右端，直到新的字符串达到指定长度len为止。
  - 如果任意参数为null时，则返回null。
  - len为负数时，返回为null。
  - pad为空串，如果len小于str长度，返回str裁剪为len长度的字符串。
- 语法
  ```
  VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
  ```
- 参数说明
  - str：起始的字符串。
  - len：新的字符串的长度。
  - pad：需要重复补充的字符串。
- 示例
  - 测试语句
    ```
    SELECT
      RPAD("adc", 2, "hello"),
      RPAD("adc", -1, "hello"),
      RPAD("adc", 10, "hello");
    ```
  - 测试结果
    ```
    "ad",,"adchellohe"
    ```

SHA1

- 功能描述

  返回字符串expr的SHA1值。
- 语法
  ```
  STRING SHA1(STRING expr)
  ```
- 参数说明
  - expr：字符串。
- 示例
  - 测试语句
    ```
    SELECT SHA1("abc");
    ```

- 测试结果
  ```
  "a9993e364706816aba3e25717850c26c9cd0d89d"
  ```

SHA256

- 功能描述

  返回字符串expr的SHA256值。
- 语法
  ```
  STRING SHA256(STRING expr)
  ```
- 参数说明
  - expr：字符串。
- 示例
  - 测试语句
    ```
    SELECT SHA256("abc");
    ```
  - 测试结果
    ```
    "ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad"
    ```

STRING_TO_ARRAY

- 功能描述

  将字符串value按delimiter分隔为字符串数组。

  说明

  delimiter使用的是java的正则表达式，若使用特殊字符则需要转义。
- 语法
  ```
  ARRAY[String] STRING_TO_ARRAY(STRING value, VARCHAR delimiter)
  ```
- 参数说明
  - value：字符串。
  - delimiter：分隔符。
- 示例
  - 测试语句
    ```
    SELECT
      string_to_array("127.0.0.1", "\\."),
      string_to_array("red-black-white-blue", "-");
    ```
  - 测试结果
    ```
    [127,0,0,1],[red,black,white,blue]
    ```

SUBSTRING

- 功能描述

  返回从指定位置开始的A的子字符串。起始位置从1开始。
  - 如果未指定len，则截取从位置start开始，到字符串结尾的子字符串。
  - 如果指定len，则截取从位置start开始，长度为len的子字符串。

  说明

  start从1开始，start为0时当1看待，为负数时表示从字符串末尾倒序计算位置。
- 语法
  ```
  VARCHAR SUBSTRING(STRING A FROM INT start)
  ```

  或


```
VARCHAR SUBSTRING(STRING A FROM INT start FOR INT len)
```

- 参数说明
  - A：指定的字符串。
  - start：在字符串A中开始截取的位置。
  - len：截取的长度。
- 示例
  - 测试语句1
    ```
    SELECT SUBSTRING("123456" FROM 2);
    ```
  - 测试结果1
    ```
    "23456"
    ```
  - 测试语句2
    ```
    SELECT SUBSTRING("123456" FROM 2 FOR 4);
    ```
  - 测试结果2
    ```
    "2345"
    ```

TRIM

- 功能描述

  从B中除去字符串首尾/首位/末尾的A。默认情况下，首尾的A都被删除。
- 语法
  ```
  STRING TRIM( { BOTH | LEADING | TRAILING } STRING a FROM STRING b)
  ```
- 参数说明
  - a：字符串。
  - b：字符串。
- 示例
  - 测试语句
    ```
    SELECT TRIM(BOTH " " FROM "  hello world  ");
    ```
  - 测试结果
    ```
    "hello world"
    ```

UPPER

- 功能描述

  返回转换为大写字符的字符串。
- 语法
  ```
  VARCHAR UPPER(A)
  ```
- 参数说明
  - A：字符串。
- 示例
  - 测试语句
    ```
    SELECT UPPER("hello world");
    ```
  - 测试结果
    ```
    "HELLO WORLD"
    ```

4.1.9.3 时间函数

Flink SQL所支持的时间函数如**表4-42**所示。

函数说明

表 4-42 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”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”。
INTERVAL string range	INTERVAL	interval表示时间间隔，有两种类型，一种为“yyyy-MM”，即保存年份和月份，精度到月份，它的Range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， 他的range可以为DAY TO HOUR，DAY TO MINUTE，DAY TO SECOND或DAY TO milliseconds，比如range为DAY TO SECOND就表示天数、小时、分钟、秒的位置都有效，精度到秒，DAY TO MINUTE就表示精度到分钟。 例如： INTERVAL '10 00:00:00.004' DAY TO milliseconds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)	INT	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，则可以使用：EXTRACT(DAY FROM DATE "2006-06-05")
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。

函数	返回值	描述
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。
QUARTER(date)	INT	从SQL日期返回一年的四分之一。
(timepoint, temporal) OVERLAPS (timepoint, temporal)	BOOLEAN	确定两个时间间隔是否重叠。时间点和时间被转换成在两个时间点（开始，结束）定义的范围之内，该计算函数是***leftEnd >= rightStart && rightEnd >= leftStart***。当左边结束时间点大于等于右边开始时间点，且右边结束时间点大于等于左边开始时间点，则函数返回true值，否则返回false。 例如： ● 左边的结束时间点是“3:55:00”（2:55:00+1:00:00）大于右边的开始点是“3:30:00”；且右边的结束时间点是“5:30:00”（3:30:00+2:00:00）大于左边开始时间点“2:55:00”，则返回值为true。 (TIME '2:55:00', INTERVAL '1' HOUR) OVERLAPS (TIME '3:30:00', INTERVAL '2' HOUR) 返回值是true。 ● 左边的结束时间点是“10:00:00”小于右边的开始点是“10:15:00”；且右边的结束时间点是“13:15:00”（10:15:00+3:00:00）大于左边开始时间点“9:00:00”，则返回值为false。 (TIME '9:00:00', TIME '10:00:00') OVERLAPS (TIME '10:15:00', INTERVAL '3' HOUR) 返回值是false。
TO_TIMESTAMP(long expr)	TIMESTAMP	将时间戳转换为时间。 该函数入参数据类型仅支持BIGINT，不支持VARCHAR，STRING等其他数据类型。 例如，TO_TIMESTAMP(1628765159000)转换后值为：2021-08-12 18:45:59。
UNIX_TIMESTAMP	BIGINT	返回指定参数的时间戳，时间戳类型为BIGINT类型，单位为“秒”。 支持如下几种使用方法： ● UNIX_TIMESTAMP()：没有参数时，返回当前时间的时间戳。 ● UNIX_TIMESTAMP(STRING datestr)：包含一个参数时，返回参数所表示的时间戳，datestr格式必须为yyyy-MM-dd HH:mm:ss。 ● UNIX_TIMESTAMP(STRING datestr, STRING format)：包含两个参数时，第二个参数可以指定datestr的格式，返回第一个参数所表示的时间戳。

函数	返回值	描述
UNIX_TIMESTAMP_MS	BIGINT	返回指定参数的时间戳，时间戳类型为BIGINT类型，单位为“毫秒”。 支持如下几种使用方法： ● UNIX_TIMESTAMP_MS()：没有参数时，返回当前时间的时间戳。 ● UNIX_TIMESTAMP_MS(STRING datestr)：包含一个参数时，返回参数所表示的时间戳，datestr格式必须为yyyy-MM-dd HH:mm:ss.SSS。 ● UNIX_TIMESTAMP_MS(STRING datestr, STRING format)：包含两个参数时，第二个参数可以指定datestr的格式，返回第一个参数所表示的时间戳。

注意事项

无。

示例

```
insert into temp SELECT Date '2015-10-11'  FROM  OrderA;//返回日期
insert into temp1 SELECT Time '12:14:50'  FROM  OrderA;//返回时间
insert into temp2 SELECT Timestamp '2015-10-11 12:14:50'  FROM  OrderA;//返回时间戳
```

4.1.9.4 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。
- Flink作业不支持使用CAST将“BIGINT”转换为“TIMESTAMP”，可以使用to_timestamp或者to_localtimestamp进行转换。

示例

将amount值转换成字符串，长度为转换后的实际长度，配置的长度无效。

```
insert into temp select cast(amount as VARCHAR(10)) from source_stream;
```

常用类型转换函数

表 4-43 常用类型转换函数

函数	说明
cast(v1 as varchar)	将v1转换为字符串类型，v1可以是数值类型，TIMESTAMP/DATE/TIME。
cast (v1 as int)	将v1转换为int, v1可以是数值类型或字符类。
cast(v1 as timestamp)	将v1转换为timestamp类型，v1可以是字符串或DATE/TIME。
cast(v1 as date)	将v1转换为date类型， v1可以是字符串或者TIMESTAMP。

- cast(v1 as varchar)
  - 测试语句：
    ```
    SELECT cast(content as varchar) FROM T1;
    ```
  - 测试数据和结果

    表 4-44 T1

content（INT）	varchar
5	"5"

- cast (v1 as int)
  - 测试语句：
    ```
    SELECT cast(content as int) FROM T1;
    ```
  - 测试数据和结果

    表 4-45 T1

content（STRING）	int
"5"	5

- cast(v1 as timestamp)
  - 测试语句：
    ```
    SELECT cast(content as timestamp) FROM T1;
    ```
  - 测试数据和结果

    表 4-46 T1

content（STRING）	timestamp
"2018-01-01 00:00:01"	1514736001000

- cast(v1 as date)
  - 测试语句：
    ```
    SELECT cast(content as date) FROM T1;
    ```
  - 测试数据和结果

    表 4-47 T1

content（TIMESTAMP）	date
1514736001000	"2018-01-01"

详细样例代码

```
/** source **/
CREATE
SOURCE STREAM car_infos (cast_int_to_varchar int, cast_String_to_int string,
case_string_to_timestamp string, case_timestamp_to_date timestamp) WITH (
  type = "dis",
  region = "xxxxx",
  channel = "dis-input",
  partition_count = "1",
  encode = "json",
  offset = "13",
  json_config =
"cast_int_to_varchar=cast_int_to_varchar;cast_String_to_int=cast_String_to_int;case_string_to_timestamp=cas
e_string_to_timestamp;case_timestamp_to_date=case_timestamp_to_date"

);
/** sink **/
CREATE
SINK STREAM cars_infos_out (cast_int_to_varchar varchar, cast_String_to_int
int, case_string_to_timestamp timestamp, case_timestamp_to_date date) WITH (
  type = "dis",
  region = "xxxxx",
  channel = "dis-output",
  partition_count = "1",
  encode = "json",
  offset = "4",
  json_config =
"cast_int_to_varchar=cast_int_to_varchar;cast_String_to_int=cast_String_to_int;case_string_to_timestamp=cas
e_string_to_timestamp;case_timestamp_to_date=case_timestamp_to_date",
  enable_output_null="true"
);
/** 统计car的静态信息 **/
INSERT
INTO
  cars_infos_out
SELECT
  cast(cast_int_to_varchar as varchar),
  cast(cast_String_to_int as int),
  cast(case_string_to_timestamp as timestamp),
  cast(case_timestamp_to_date as date)
FROM
  car_infos;
```

返回数据

```
{"case_string_to_timestamp":1514736001000,"cast_int_to_varchar":"5","case_timestamp_to_date":"2018-01-
01","cast_String_to_int":100}
```

4.1.9.5 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4-48所示。

示例数据：表T1

```
|score|
|81   |
|100  |
|60   |
|95   |
|86   |
```

常用聚合函数

表 4-48 常用聚合函数表

函数	返回值类型	描述
COUNT(*)	BIGINT	返回元组个数。
COUNT([ALL] expression...)	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。
AVG(numeric)	DOUBLE	返回所有输入值的数字的平均值（算术平均值）。
SUM(numeric)	DOUBLE	返回所有输入值之间的数值之和。
MAX(value)	DOUBLE	返回所有输入值的值的最大值。
MIN(value)	DOUBLE	返回所有输入值的值的最小值。
STDDEV_POP(value)	DOUBLE	返回所有输入值之间的数字字段的总体标准偏差。
STDDEV_SAMP(value)	DOUBLE	返回所有输入值之间的数字字段的样本标准偏差。
VAR_POP(value)	DOUBLE	返回所有输入值之间的数字字段的总体方差（总体标准偏差的平方）。
VAR_SAMP(value)	DOUBLE	返回所有输入值之间的数字字段的样本方差（样本标准偏差的平方）。

示例

- COUNT(*)
  - 测试语句：
    ```
    SELECT COUNT(score) FROM T1;
    ```
  - 测试数据和结果

表 4-49 T1

测试数据（score）	测试结果
81	5
100	
60	
95	
86	

- COUNT([ALL] expression | DISTINCT expression1 [, expression2]*)
  - 测试语句：
    ```
    SELECT COUNT(DISTINCT content ) FROM T1;
    ```
  - 测试数据和结果

表 4-50 T1

content (STRING)	测试结果
"hello1 "	2
"hello2 "	
"hello2"	
null	
86	

- AVG(numeric)
  - 测试语句：
    ```
    SELECT AVG(score) FROM T1;
    ```
  - 测试数据和结果

表 4-51 T1

测试数据（score）	测试结果
81	84.0
100	
60	
95	
86	

- SUM(numeric)
  - 测试语句：


```
SELECT SUM(score) FROM T1;
```

- 测试数据和结果

表 4-52 T1

测试数据（score）	测试结果
81	422.0
100	
60	
95	
86	

- MAX(value)
  - 测试语句：
    ```
    SELECT MAX(score) FROM T1;
    ```
  - 测试数据和结果

表 4-53 T1

测试数据（score）	测试结果
81	100.0
100	
60	
95	
86	

- MIN(value)
  - 测试语句：
    ```
    SELECT MIN(score) FROM T1;
    ```
  - 测试数据和结果

表 4-54 T1

测试数据（score）	测试结果
81	60.0
100	
60	
95	
86	

- STDDEV_POP(value)
  - 测试语句：
    ```
    SELECT STDDEV_POP(score) FROM T1;
    ```
  - 测试数据和结果

    表 4-55 T1

测试数据（score）	测试结果
81	13.0
100	
60	
95	
86	

- STDDEV_SAMP(value)
  - 测试语句：
    ```
    SELECT STDDEV_SAMP(score) FROM T1;
    ```
  - 测试数据和结果

    表 4-56 T1

测试数据（score）	测试结果
81	15.0
100	
60	
95	
86	

- VAR_POP(value)
  - 测试语句：
    ```
    SELECT VAR_POP(score) FROM T1;
    ```
  - 测试数据和结果

    表 4-57 T1

测试数据（score）	测试结果
81	193.0
100	
60	
95	

测试数据（score）	测试结果
86	

- VAR_SAMP(value)
  - 测试语句：
    ```
    SELECT VAR_SAMP(score) FROM T1;
    ```
  - 测试数据和结果

    表 4-58 T1

测试数据（score）	测试结果
81	241.0
100	
60	
95	
86	

4.1.9.6 表值函数

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 4-59 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 source stream 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;
```

4.1.9.7 其他函数

数组函数

表 4-60 数组函数表

函数	返回值类型	描述
CARDINALITY(ARRAY)	INT	返回数组的元素个数。
ELEMENT(ARRAY)	-	使用单个元素返回数组的唯一元素。 如果数组为空，则返回null。如果数组有多个元素，则抛出异常。

示例：

返回数组的元素个数为3。

```
insert into temp select CARDINALITY(ARRAY[TRUE, TRUE, FALSE]) from source_stream;
```

返回'HELLO WORLD'。

```
insert into temp select ELEMENT(ARRAY['HELLO WORLD']) from source_stream;
```

属性访问函数

表 4-61 属性访问函数表

函数	返回值类型	描述
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。

4.1.10 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_2.11</artifactId>
  <version>1.7.2</version>
  <scope>provided</scope>
</dependency>
<dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-streaming-java_2.11</artifactId>
      <version>1.7.2</version>
      <scope>provided</scope>
</dependency>
```

注意事项

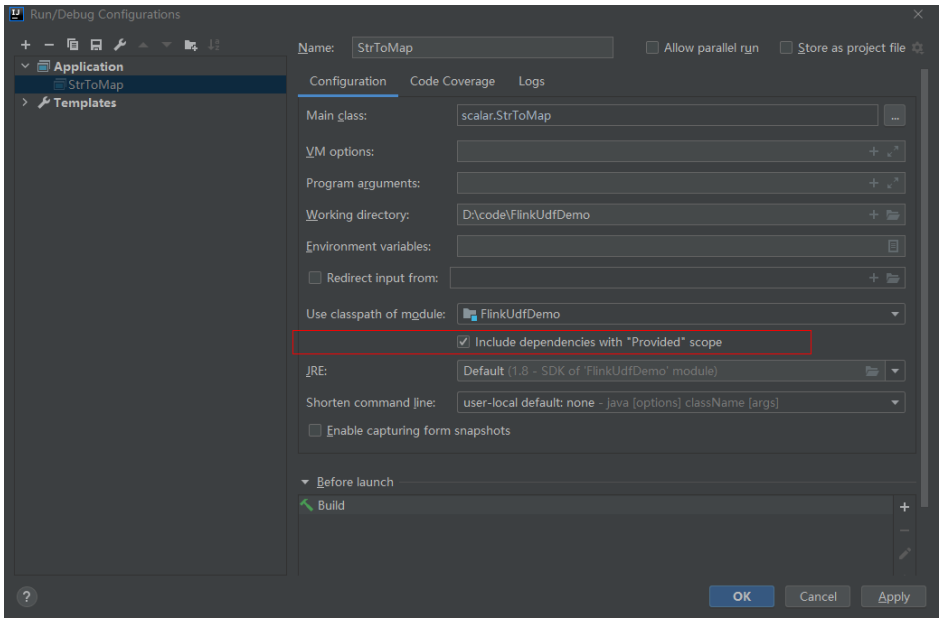
- **暂不支持通过python写UDF、UDTF、UDAF自定义函数。**
- 如果使用IntelliJ IDEA工具对创建的自定义函数进行调试，则需要在IDEA上勾选：include dependencies with "Provided" scope，否则本地调试运行时会加载不到pom文件中的依赖包。

  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。

c. 单击“OK”完成应用配置。

使用方式

1. 编写自定义函数代码。具体的代码样例可以参考**UDF**、**UDTF**或者**UDAF**。
2. 将写好的自定义函数编译并打成JAR包，并上传到OBS上。
3. 在DLI管理控制台的左侧导航栏中，单击“作业管理”>“Flink作业”，在需要编辑的Flink SQL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”页签中，“所属队列”选择专享队列，会出现“UDF Jar”参数，在此处选择存放在OBS上的JAR文件，单击“保存”。

   说明

   在选择自定义函数Jar包之前需要将对应的jar包上传至已创建好的OBS桶中。

   选定JAR包以后，在SQL里添加UDF声明语句，就可以像普通函数一样使用了。

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.xxx.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.xxx.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.xxx.udf.UdfAggFunction';
INSERT INTO sink_stream SELECT udaf_test(attr2) FROM source_stream GROUP BY attr1;
```

4.1.11 地理函数

函数说明

基本地理空间几何元素介绍说明如**表4-62**所示。

表 4-62 基本地理空间几何元素表

地理空间几何元素（统称geometry）	说明	举例
ST_POINT(latitude, longitude)	地理点，包含经度和维度两个信息。	ST_POINT(1.12012, 1.23401)
ST_LINE(array[point1...pointN])	地理线，由多个地理点（ST_POINT）按顺序连接成的折线或直线。	ST_LINE(ARRAY[ST_POINT(1.12, 2.23), ST_POINT(1.13, 2.44), ST_POINT(1.13, 2.44)])
ST_POLYGON(array[point1...point1])	地理多边形，由首尾相同的多个地理点（ST_POINT）按顺序连线围成的封闭多边形区域。	ST_POLYGON(ARRAY[ST_POINT(1.0, 1.0), ST_POINT(2.0, 1.0), ST_POINT(2.0, 2.0), ST_POINT(1.0, 1.0)])
ST_CIRCLE(point, radius)	地理圆形，由圆心地理点（ST_POINT）和半径构成的地理圆形区域。	ST_CIRCLE(ST_POINT(1.0, 1.0), 1.234)

用户可以以基本地理空间几何元素为基础，构造复杂的地理空间几何元素，具体的变换方法见表4-63。

表 4-63 基于基本地理空间几何元素构造复杂几何元素的变换表

变换方法	说明	举例
ST_BUFFER(geometry, distance)	创建一个环绕包含给定地理空间几何元素的多边形，并以给定距离作为环绕距离，通常使用该函数构造一定宽度的公路范围用于偏航检测。	ST_BUFFER(ST_LINE(ARRAY[ST_POINT(1.12, 2.23), ST_POINT(1.13, 2.44), ST_POINT(1.13, 2.44)]),1.0)
ST_INTERSECTION(geometry, geometry)	创建一个多边形，其范围为给定的两个地理空间几何元素的交叠区域。	ST_INTERSECTION(ST_CIRCLE(ST_POINT(1.0, 1.0), 2.0), ST_CIRCLE(ST_POINT(3.0, 1.0), 1.234))
ST_ENVELOPE(geometry)	创建一个包含给定的地理空间几何元素的最小矩形。	ST_ENVELOPE(ST_CIRCLE(ST_POINT(1.0, 1.0), 2.0))

DLI提供丰富的对地理空间几何元素的操作和位置判断函数，具体的SQL标量函数介绍说明见表4-64。

表 4-64 SQL 标量函数表

函数	返回值	说明
ST_DISTANCE(point_1, point_2)	DOUBLE	计算两个地理点之间的欧几里得距离。 示例如下： Select ST_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input
ST_GEODESIC_DISTANCE(point_1, point_2)	DOUBLE	计算两个地理点之间的测地距离，即两个地理点之间地表最短路径距离。 示例如下： Select ST_GEODESIC_DISTANCE(ST_POINT(x1, y1), ST_POINT(x2, y2)) FROM input
ST_PERIMETER(polygon)	DOUBLE	计算多边形的周长。 示例如下： Select ST_PERIMETER(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]) FROM input
ST_AREA(polygon)	DOUBLE	计算多边形区域的面积。 示例如下： Select ST_AREA(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]) FROM input
ST_OVERLAPS(polygon_1, polygon_2)	BOOLEAN	判断一个多边形是否与另一个多边形有重叠区域。 示例如下： SELECT ST_OVERLAPS(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input

函数	返回值	说明
ST_INTERSECT(line1, line2)	BOOLEAN	检查两条线段是否相互交叉，而非线条所在的直线是否交叉。 示例如下： SELECT ST_INTERSECT(ST_LINE(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12)]), ST_LINE(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23)])) FROM input
ST_WITHIN(point, polygon)	BOOLEAN	一个点是否包含在几何体（多边形或圆形）内。 示例如下： SELECT ST_WITHIN(ST_POINT(x11, y11), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_CONTAINS(polygon_1, polygon_2)	BOOLEAN	判断第一个几何体是否包含第二个几何体。 示例如下： SELECT ST_CONTAINS(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_COVERS(polygon_1, polygon_2)	BOOLEAN	第一个几何体是否覆盖第二个几何体。与ST_CONTAINS相似，但在边界重叠情况下ST_COVER判断为TRUE，ST_CONTAINS判断为FALSE。 示例如下： SELECT ST_COVERS(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON([ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input
ST_DISJOINT(polygon_1, polygon_2)	BOOLEAN	判断一个多边形是否与另一个多边形不相交（不重叠）。 示例如下： SELECT ST_DISJOINT(ST_POLYGON(ARRAY[ST_POINT(x11, y11), ST_POINT(x12, y12), ST_POINT(x11, y11)]), ST_POLYGON(ARRAY[ST_POINT(x21, y21), ST_POINT(x22, y22), ST_POINT(x23, y23), ST_POINT(x21, y21)])) FROM input

地理函数的基准坐标系标准为全球通用的GPS坐标系标准WGS84，GPS坐标不能直接在百度地图（BD09标准）或者google地图（GCJ02标准）上使用，会有偏移现象，为了在不同地理坐标系之间切换，DLI提供了坐标系转换的一系列函数，并且还提供地理距离与米之间的转换函数。详见**表4-65**。

表 4-65 地理坐标系转换函数与距离单位转换函数表

函数	返回值	说明
WGS84_TO_BD09(geometry)	对应的百度地图坐标系地理空间几何元素	将GPS坐标系下的地理空间几何元素转换成百度地图坐标系下对应的地理空间几何元素。示例如下： WGS84_TO_BD09(ST_CIRCLE(ST_POINT(x, y), r))
WGS84_TO_CJ02(geometry)	对应的Google地图坐标系地理空间几何元素	将GPS坐标系下的地理空间几何元素转换成Google地图坐标系下对应的地理空间几何元素。示例如下： WGS84_TO_CJ02(ST_CIRCLE(ST_POINT(x, y), r))
BD09_TO_WGS84(geometry)	对应的GPS坐标系地理空间几何元素	将百度地图坐标系下的地理空间几何元素转换成GPS坐标系下对应的地理空间几何元素。示例如下： BD09_TO_WGS84(ST_CIRCLE(ST_POINT(x, y), r))
BD09_TO_CJ02(geometry)	对应的Google地图坐标系地理空间几何元素	将百度地图坐标系下的地理空间几何元素转换成Google地图坐标系下对应的地理空间几何元素。示例如下： BD09_TO_CJ02(ST_CIRCLE(ST_POINT(x, y), r))
CJ02_TO_WGS84(geometry)	对应的GPS坐标系地理空间几何元素	将Google地图坐标系下的地理空间几何元素转换成GPS坐标系下对应的地理空间几何元素。示例如下： CJ02_TO_WGS84(ST_CIRCLE(ST_POINT(x, y), r))

函数	返回值	说明
CJ02_TO_BD09(geometry)	对应的百度地图坐标系地理空间几何元素	将Google地图坐标系下的地理空间几何元素转换成百度地图坐标系下对应的地理空间几何元素。示例如下： CJ02_TO_BD09(ST_CIRCLE(ST_POINT(x, y), r))
DEGREE_TO_METER(distance)	DOUBLE	将地理函数的距离数值转换成以“米”为单位的数值。示例如下（以米为单位计算地理三边形周长）： DEGREE_TO_METER(ST_PERIMETER(ST_POLYGON(ARRAY[ST_POINT(x1,y1), ST_POINT(x2,y2), ST_POINT(x3,y3), ST_POINT(x1,y1)])))
METER_TO_DEGREE(numerical_value)	DOUBLE	将以“米”为单位的数值转换成地理函数可计算的距离单位数值。示例如下（画出以指定地理点为圆心，半径1公里的圆）： ST_CIRCLE(ST_POINT(x,y), METER_TO_DEGREE(1000))

DLI还提供了基于窗口的SQL地理聚合函数用于SQL逻辑涉及窗口和聚合的场景。详见表4-66的介绍说明。

表 4-66 时间相关 SQL 地理聚合函数表

函数	说明	举例
AGG_DISTANCE(point)	距离聚合函数，用于计算窗口内所有相邻地理点的距离总和。	SELECT AGG_DISTANCE(ST_POINT(x,y)) FROM input GROUP BY HOP(rowtime, INTERVAL '1' HOUR, INTERVAL '1' DAY)

函数	说明	举例
AVG_SPEED(point)	平均速度聚合函数，用于计算窗口内所有地理点组成的移动轨迹的平均速度，单位为“米/秒”。	SELECT AVG_SPEED(ST_POINT(x,y)) FROM input GROUP BY TUMBLE(proctime, INTERVAL '1' DAY)

注意事项

无。

示例

偏航检测样例：

```
INSERT INTO yaw_warning
SELECT "The car is yawing"
FROM driver_behavior
WHERE NOT ST_WITHIN(ST_POINT(cast(Longitude as DOUBLE), cast(Latitude as DOUBLE)),
ST_BUFFER(ST_LINE(ARRAY[ST_POINT(34.585555,105.725221),ST_POINT(34.586729,105.735974),ST_POINT(
34.586492,105.740538),ST_POINT(34.586388,105.741651),ST_POINT(34.586135,105.748712),ST_POINT(34.5
88691,105.74997)]),0.001));
```

IP 地理函数

说明

当前仅支持IPV4的IP地址。

表 4-67 IP 地理函数表

函数	返回值	说明
IP_TO_COUNTRY	STRING	获取IP地址所在的国家名称。
IP_TO_PROVINCE	STRING	获取IP地址所在的省份。 用法说明： ● IP_TO_PROVINCE(STRING ip)：返回IP地址所在的省份。 ● IP_TO_PROVINCE(STRING ip, STRING lang)：以指定语言返回IP地址所在的省份。 **说明** ● 当IP无法被解析到省份时，返回该IP所属的国家。当IP无法被解析时，返回“未知”。 ● 函数返回的省份名称均为简称。

函数	返回值	说明
IP_TO_CITY	STRING	获取IP地址所在的城市名称。 **说明** 当IP无法被解析到城市时，返回该IP所属的省份或者国家。当IP无法被解析时，返回“未知”。
IP_TO_CITY_GEO	STRING	获取IP地址所在城市的经纬度，格式为“纬度,经度”。 用法说明： IP_TO_CITY_GEO(STRING ip)：返回IP所在城市的经纬度。

4.1.12 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或者插入常量数据。

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

```
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

无。

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;
```

用表达式 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);
```

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;
```

UNION

语法格式

```
query UNION [ ALL ] query
```

语法说明

UNION返回多个查询结果的并集。

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。

```
insert into temp SELECT  * FROM Orders1
  UNION SELECT  * FROM Orders2;
```

4.1.13 条件表达式

CASE 表达式

语法格式

```
CASE value WHEN value1 [, value11 ]* THEN result1
  [ WHEN valueN [, valueN1 ]* THEN resultN ]* [ ELSE resultZ ]
  END
```

或

```
CASE WHEN condition1 THEN result1
  [ WHEN conditionN THEN resultN ]* [ ELSE resultZ ]
  END
```

语法说明

- 当value值为value1则返回result1，不满足则返回resultZ，若没有else语句，则返回null。

- 当condition1为true时返回result1，不满足则返回resultZ，若没有else语句，则返回null。

注意事项

- 所有result的类型都必须一致。
- 所有condition类型都必须是布尔类型。
- 当没有满足的分支时，若指定else语句，则返回else的值，若没有else语句，则返回null。

示例

当units等于5时返回1，否则返回0。

示例1：

```
insert into temp SELECT  CASE units WHEN 5 THEN 1 ELSE 0 END FROM Orders;
```

示例2：

```
insert into temp SELECT CASE WHEN units = 5 THEN 1 ELSE 0 END FROM Orders;
```

NULLIF 表达式

语法格式

```
NULLIF(value, value)
```

语法说明

如果值相同，则返回NULL。 例如，NULLIF（5，5）返回NULL; NULLIF（5，0）返回5。

注意事项

无。

示例

当units等于3时返回null，否则返回units。

```
insert into temp SELECT  NULLIF(units, 3) FROM Orders;
```

COALESCE 表达式

语法格式

```
COALESCE(value, value [, value ]* )
```

语法说明

返回从左到右第一个不为NULL的参数的值。

注意事项

所有value的类型都必须一致。

示例

返回5。

```
insert into temp SELECT  COALESCE(NULL, 5) FROM Orders;
```

4.1.14 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：

说明

- time_attr可以设置processing-time或者event-time。
  - time_attr设置为event-time时参数类型为bigint或者timestamp类型。
  - time_attr设置为processing-time时无需指定类型。
- interval设置窗口周期。

- 分组函数

表 4-68 分组函数表

函数名	说明
TUMBLE(time_attr, interval)	跳跃窗口。
HOP(time_attr, interval, interval)	拓展的跳跃窗口(等价于datastream的滑动窗口)，可以分别设置输出触发周期和窗口周期。
SESSION(time_attr, interval)	会话窗口，interval表示多长时间没有记录则关闭窗口。

- 窗口函数

表 4-69 窗口函数表

函数名	说明
TUMBLE_START(time_attr, interval)	返回跳跃窗口开始时间。为UTC时区。
TUMBLE_END(time_attr, interval)	返回跳跃窗口结束时间。为UTC时区。
HOP_START(time_attr, interval, interval)	返回拓展的跳跃窗口开始时间。为UTC时区。
HOP_END(time_attr, interval, interval)	返回拓展的跳跃窗口结束时间。为UTC时区。
SESSION_START(time_attr, interval)	返回会话窗口开始时间。为UTC时区。
SESSION_END(time_attr, interval)	返回会话窗口结束时间。为UTC时区。

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;
```

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

```
OVER (
  [PARTITION BY partition_name]
  ORDER BY proctime|rowtime(ROWS number PRECEDING) |(RANGE (BETWEEN INTERVAL '1' SECOND
PRECEDING AND CURRENT ROW | UNBOUNDED preceding))
)
```

语法说明

表 4-70 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

- 同一select里所有聚合函数定义的窗口都必须保持一致。
- 当前Over窗口只支持前向计算(preceding)，不支持following计算。

- 必须指定ORDER BY 按processing time或event time。
- 不支持对常量做聚合操作，如sum(2)。

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;
```

4.1.15 流表 JOIN

流与表进行连接操作，从表中查询并补全流字段。目前支持连接RDS表和DCS服务的Redis表。通过ON条件描述查询的Key，并补全表结构的Value字段。

RDS表的数据定义语句请参见**创建RDS表**。

Redis表的数据定义语句请参见**创建Redis表**。

语法格式

```
FROM tableExpression JOIN tableExpression
  ON value11 = value21 [ AND value12 = value22]
```

语法说明

ON条件中只支持表属性等值查询，当存在二级Key时（Redis值类型为HASH情况下），需要AND表达Key和Hash Key等值查询。

注意事项

无。

示例

将车辆信息输入流与车辆价格表做等值连接后，获取车辆价格信息并填入车辆信息输出流后输出。

```
CREATE SOURCE STREAM car_infos (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_detail_type STRING
```

```
)
WITH (
  type = "dis",
  region = "",
  channel = "dliinput",
  partition_count = "1",
  encode = "csv",
  field_delimiter = ","
);

/** 创建数据维表，用于和输入流连接，实现字段回填
  *
  * 根据实际情况修改以下选项：
  * value_type：redis的键值对应值类型，支持STRING、HASH、SET、ZSET、LIST，其中HASH类型需要指定
hash_key_column作为二层主键，集合类型将用逗号拼接所有查询出来的值
  * key_column：维表主键对应的列名
  * hash_key_column：当redis的键值对应值类型为HASH时，HASHMAP的KEY对应的列名，当值类型非HASH
时，无需指定该配置
  * cluster_address：DCS服务redis集群地址
  * password：DCS服务redis集群密码
  **/
CREATE TABLE car_price_table (
  car_brand STRING,
  car_detail_type STRING,
  car_price STRING
)
WITH (
  type = "dcs_redis",
  value_type = "hash",
  key_column = "car_brand",
  hash_key_column = "car_detail_type",
  cluster_address = "192.168.1.238:6379",
  password = "xxxxxxxx"
);

CREATE SINK STREAM audi_car_owner_info (
  car_id STRING,
  car_owner STRING,
  car_brand STRING,
  car_detail_type STRING,
  car_price STRING
)
WITH (
  type = "dis",
  region = "",
  channel = "dlioutput",
  partition_key = "car_owner",
  encode = "csv",
  field_delimiter = ","
);

INSERT INTO audi_car_owner_info
SELECT t1.car_id, t1.car_owner, t2.car_brand, t1.car_detail_type, t2.car_price
FROM car_infos as t1 join car_price_table as t2
ON t2.car_brand = t1.car_brand and t2.car_detail_type = t1.car_detail_type
WHERE t1.car_brand = "audi";
```

4.1.16 配置时间模型

Flink中主要提供两种时间模型：Processing Time和Event Time。

DLI允许在创建Source Stream和Temp Stream的时候指定时间模型以便在后续计算中使用。

配置 Processing Time

Processing Time是指系统时间，与数据本身的时间戳无关，即在Flink算子内计算完成的时间。

语法格式

```
CREATE SOURCE STREAM stream_name(...) WITH (...)
TIMESTAMP BY proctime.proctime;
CREATE TEMP STREAM stream_name(...)
TIMESTAMP BY proctime.proctime;
```

语法说明

设置Processing Time只需在timestamp by后配置proctime.proctime即可，后续可以直接使用proctime字段。

注意事项

无。

示例

```
CREATE SOURCE STREAM student_scores (
  student_number STRING, /* 学号 */
  student_name STRING, /* 姓名 */
  subject STRING, /* 学科 */
  score INT /* 成绩 */
)
WITH (
  type = "dis",
  region = "",
  channel = "dliinput",
  partition_count = "1",
  encode = "csv",
  field_delimiter=","
)TIMESTAMP BY proctime.proctime;

INSERT INTO score_greate_90
SELECT student_name, sum(score) over (order by proctime RANGE UNBOUNDED PRECEDING)
FROM student_scores;
```

配置 Event Time

Event Time是指事件产生的时间，即数据产生时自带时间戳。

语法格式

```
CREATE SOURCE STREAM stream_name(...) WITH (...)
TIMESTAMP BY {attr_name}.rowtime
SET WATERMARK (RANGE {time_interval} | ROWS {literal}, {time_interval});
```

语法说明

设置Event Time需要选定流中的某一个属性来作为时间戳，同时需要设置Watermark策略。

由于网络等原因，有时会导致乱序的产生；对于迟来的数据，需要Watermark来保证一个特定的时间后去触发Window进行计算。Watermark主要是用来处理乱序数据，流处理从事件产生，到发送到DLI服务，中间有一个过程。

Watermark有两种设置策略：

- 按时间周期
  ```
  SET WATERMARK(range interval {time_unit}, interval {time_unit})
  ```

- 按事件个数
  ```
  SET WATERMARK(rows literal, interval {time_unit})
  ```

  说明

  一个逗号表示一个参数，第一个参数表示Watermark发送周期，第二个参数表示允许最大延迟时间。

注意事项

无。

示例

- time2事件产生时间开始，每10s发送一次watermark，事件最大允许延迟时间为20s。
  ```
  CREATE SOURCE STREAM student_scores (
    student_number STRING, /* 学号 */
    student_name STRING, /* 姓名 */
    subject STRING, /* 学科 */
    score INT, /* 成绩 */
    time2 TIMESTAMP
  )
  WITH (
    type = "dis",
    region = "",
    channel = "dliinput",
    partition_count = "1",
    encode = "csv",
    field_delimiter=","
  )
  TIMESTAMP BY time2.rowtime
  SET WATERMARK (RANGE interval 10 second, interval 20 second);

  INSERT INTO score_greate_90
  SELECT student_name, sum(score) over (order by time2 RANGE UNBOUNDED PRECEDING)
  FROM student_scores;
  ```
- 每收到10个数据发送一次watermark，事件最大允许延迟时间为20s。
  ```
  CREATE SOURCE STREAM student_scores (
    student_number STRING, /* 学号 */
    student_name STRING, /* 姓名 */
    subject STRING, /* 学科 */
    score INT, /* 成绩 */
    time2 TIMESTAMP
  )
  WITH (
    type = "dis",
    region = "",
    channel = "dliinput",
    partition_count = "1",
    encode = "csv",
    field_delimiter=","
  )
  TIMESTAMP BY time2.rowtime
  SET WATERMARK (ROWS 10, interval 20 second);

  INSERT INTO score_greate_90
  SELECT student_name, sum(score) over (order by time2 RANGE UNBOUNDED PRECEDING)
  FROM student_scores;
  ```

4.1.17 CEP 模式匹配

复杂事件处理（Complex Event Process，简称CEP）用来检测无尽数据流中的复杂模式，拥有从不同的数据行中辨识查找模式的能力。模式匹配是复杂事件处理的一个强大援助。

例子包括受一系列事件驱动的各种业务流程，例如在安全应用中侦测异常行为；在金融应用中查找价格、交易量和其他行为的模式。其他常见的用途如欺诈检测应用和传感器数据的分析等。

语法格式

```
MATCH_RECOGNIZE (
    [ PARTITION BY expression [, expression ]* ]
    [ ORDER BY orderItem [, orderItem ]* ]
    [ MEASURES measureColumn [, measureColumn ]* ]
    [ ONE ROW PER MATCH | ALL ROWS 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 ]*
) MR
```

说明

SQL中的模式匹配是用MATCH_RECOGNIZE子句执行。MATCH_RECOGNIZE子句执行如下任务：

- 使用PARTITION BY 和ORDER BY子句对MATCH_RECOGNIZE子句中的数据进行逻辑分区和排序。
- 使用PATTERN子句来定义要查找的数据行的模式。这些模式使用规则表达式语法。
- 使用DEFINE子句指定PATTERN模式变量所需的逻辑条件。
- 使用MEASURES子句定义度量，这是一些可在SQL查询的其他部分所使用的表达式。

语法说明

表 4-71 语法说明

参数	是否必选	说明
PARTITION BY	否	将数据行进行逻辑上的分组。
ORDER BY	否	在分区中对数据行进行逻辑排序。

参数	是否必选	说明
[ONE ROW | ALL ROWS] PER MATCH	否	为每个匹配选择输出汇总或者明细。 ● ONE ROW PER MATCH：每次检测到完整的匹配后进行汇总输出。 ● ALL ROWS PER MATCH：检测到完整的匹配后会把匹配过程中每条具体记录进行输出。 示例如下： `SELECT * FROM MyTable MATCH_RECOGNIZE ( MEASURES AVG(B.id) as Bid ALL ROWS PER MATCH PATTERN (A B C) DEFINE A AS A.name = 'a', B AS B.name = 'b', C as C.name = 'c' ) MR` 示例说明： 假设MyTable数据格式为(id,name), 有三条数据(1,a) (2,b), (3,c)。 那么ONE ROW PER MATCH会输出B的平均值2。 ALL ROWS PER MATCH会将每条记录及B的平均值输出，也就是输出(1,a, null), (2,b,2), (3,c,2)。
MEASURES	否	定义要输出的度量值。
PATTERN	是	定义要匹配的模式。 ● 连续事件 PATTERN (A B C)即表示检测连续的ABC事件。 ● 逻辑事件PATTERN (A | B)即表示检测A或者B。 ● 修饰符 – *： 0次或多次迭代，如A* 匹配0次或多次A – +： 1次或多次迭代，如A+ 匹配1次或多次A – ?： 0次或1次迭代，如A? 匹配0次或多次A – {n}： n 次迭代 (n > 0)，如A{5} 匹配5次A – {n,}： n 次或更多次迭代 (n >= 0)，如A{5,} 匹配>=5次A – {n,m}： n次至m次 (包括n和m) 迭代 (0 <= n <= m, 0 < m)，如A{3,6} 匹配3至6次A – {,m}： 0次至m次 (包括0和m) 迭代 (m > 0)，如A{,4} 匹配0至4次A
DEFINE	是	定义主要的模式变量条件。

参数	是否必选	说明
AFTER MATCH SKIP	否	定义在一个匹配找到之后从哪里开始下一轮匹配。 • SKIP TO NEXT ROW ： 在当前匹配第一行之后的下一行开始下一轮模式匹配 • SKIP PAST LAST ROW ： 在当前匹配的最后一行之后的下一行开始下一轮匹配 • SKIP TO FIRST variable：从当前匹配的第一个variable开始下一轮匹配 • SKIP TO LAST variable：从当前匹配的最后一个variable开始下一轮匹配 • SKIP TO variable：同SKIP TO LAST variable

CEP 支持的函数

表 4-72 函数说明

函数	说明
MATCH_NUMBER()	说明本次匹配属于第几次匹配。可用在MEASURES和DEFINE子句中。
CLASSIFIER()	说明当前记录被匹配到PATTERN里的哪个模式变量里。可用在MEASURES和DEFINE子句中。
FIRST()/LAST()	每次匹配里的第一个/最后一个记录。比如PATTERN (A B+ C), FIRST(B.id)代表匹配里的第一个B的id，LAST(B.id)代表匹配里的最后一个B的id。
NEXT()/PREV()	相对偏移，可用在DEFINE里。比如PATTERN (A B+) DEFINE B AS B.price > PREV(B.price)。
RUNNING/ FINAL	RUNNING 表示匹配过程中间值，FINAL表示最终结果值，RUNNING/FINAL一般只在ALL ROWS PER MATCH里才有意义。比如有三条记录(a, 2), (b, 6), (c, 12), 那么RUNNING AVG(A.price)和FINAL AVG(A.price)的值就是(2, 6), (4, 6), (6, 6)。
聚合函数(COUNT, SUM, AVG, MAX, MIN)	聚合操作，可用在MEASURES和DEFINE子句中。关于聚合函数的详细信息，请参见**聚合函数**。

示例

- 套牌车检测

5分钟内在不同区域的城市道路或者高速道路的摄像头采集到相同牌照的车辆数据，通过对号牌切换特征的模式匹配，实现套牌车检测。

```
INSERT INTO fake_licensed_car
SELECT * FROM camera_license_data MATCH_RECOGNIZE
(
  PARTITION BY car_license_number
  ORDER BY proctime
  MEASURES A.car_license_number as car_license_number, A.camera_zone_number as first_zone,
B.camera_zone_number as second_zone
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST C
  PATTERN (A B+ C)
  WITHIN interval '5' minute
  DEFINE
    B AS B.camera_zone_number <> A.camera_zone_number,
    C AS C.camera_zone_number = A.camera_zone_number
) MR;
```

该规则表示5分钟内在两个不同摄像区域内检测到同一车牌号车辆，为了防止出现误判，即车辆确实从A区域行驶到B区域，检查到B区域后A区域又检测到了该车牌，这种情况则认为是真正的套牌车。

输入数据：

```
浙B88888,zone_A
浙AZ626M,zone_A
浙B88888,zone_A
浙AZ626M,zone_A
浙AZ626M,zone_A
浙B88888,zone_B
浙B88888,zone_B
浙AZ626M,zone_B
浙AZ626M,zone_B
浙AZ626M,zone_C
浙B88888,zone_A
浙B88888,zone_A
```

则会输出：

```
浙B88888,zone_A,zone_B
```

4.1.18 StreamingML

4.1.18.1 异常检测

异常检测应用场景相当广泛，包括了入侵检测，金融诈骗检测，传感器数据监控，医疗诊断和自然数据检测等。异常检测经典算法包括统计建模方法，基于距离计算方法，线性模型和非线性模型等。

我们采用一种基于随机森林的异常检测方法：

- One-pass算法，O(1)均摊时空复杂度。
- 随机森林结构仅构造一次，模型更新仅仅是节点数据分布值的更新。
- 节点存储多个窗口的数据分布信息，能够检测数据分布变化。
- 异常检测和模型更新在同一个代码框架中完成。

语法格式

```
SRF_UNSUP(ARRAY[字段1, 字段2, ...], '可选参数列表')
```

说明

- 函数输出为[0, 1]区间的DOUBLE值，表示数据的异常打分。
- 字段名必须为一致的数值类型，若字段类型不同，可通过CAST函数转义，例如[a, CAST(b as DOUBLE)]。
- 可选参数列表语法为"key1=value,key2=value2,..."。

参数说明

表 4-73 参数说明

参数	是否必选	说明	默认值
transientThreshold	否	连续transientThreshold个窗口发生数据改变表示发生数据概念迁移。	5
numTrees	否	随机森林中Tree的数量。	15
maxLeafCount	否	Tree最大叶子节点数量。	15
maxTreeHeight	否	Tree最大高度。	12
seed	否	算法使用的随机种子值。	4010
numClusters	否	分类数，默认包含异常和非异常两类。	2
dataViewMode	否	算法学习模式。 ● history：学习所有历史数据。 ● horizon：仅考虑最近一段时间历史数据，默认为4个窗口。	history

示例

对于数据流MyTable中的c字段运行异常检测算法，当异常分大于0.8时输出异常。

```
SELECT c,
   CASE WHEN SRF_UNSUP(ARRAY[c], "numTrees=15,seed=4010") OVER (ORDER BY proctime RANGE
BETWEEN INTERVAL '99' SECOND PRECEDING AND CURRENT ROW) > 0.8
        THEN 'anomaly'
        ELSE 'not anomaly'
   END
FROM MyTable
```

4.1.18.2 时间序列预测

流数据处理中经常需要对于时间序列数据进行建模和预测，建模是指提取数据中有用的统计信息和数据特征，预测是指使用模型对未来的数据进行推测。DLI服务提供了一系列随机线性模型，帮助用户在线实时进行模型的建模和预测。

ARIMA (Non-Seasonal)

ARIMA（Auto-Regressive Integrated Moving Average）是时间序列预测中的经典模型，和AR/MA/ARMA模型之间联系紧密。

- AR/MA/ARMA适用于**平稳**序列 (stationary)
  - AR(p)：自回归模型，当前值可以描述为p个之前值的线性组合。利用线性组合的权值即可预测下一个值。
  - MA(q)：移动平均模型，当前值可以描述为序列均值加上q个之前值的白噪声的线性组合。利用线性组合的权值也可预测下一个值。
  - ARMA(p, q)：自回归移动平均模型，综合了AR和MA两个模型的优势，在ARMA模型中，自回归过程负责量化当前数据与前期数据之间的关系，移动平均过程负责解决随机变动项的求解问题，因此，该模型比AR/MA更为有效和常用。
- ARIMA适用于**非平稳**序列 (non-stationary)。ARIMA(p, q, d)中p为自回归项数，q为滑动平均项数，d为使之成为平稳序列所做的差分次数（阶数）。

语法格式

```
AR_PRED(field, degree)：使用AR模型预测新数据。
AR_COEF(field, degree)：返回AR模型的权值。
ARMA_PRED(field, degree)：使用ARMA模型预测新数据。
ARMA_COEF(field, degree)：返回ARMA模型的权值。
ARIMA_PRED(field, degree, derivativeOrder)：使用ARIMA预测新数据。
```

表 4-74 参数说明

参数	是否必选	说明	默认值
field	是	数据在数据流中的字段名。	-
degree	否	指定使用之前数据项的个数，当前实现中限定p = q = degree。	5
derivativeOrder	否	指定差分次数，通常设置为1或者2。	1

示例

分别使用AR，ARMA，ARIMA结合窗口进行时间序列预测。

```
SELECT b,
   AR_PRED(b) OVER (ORDER BY rowtime ROWS  BETWEEN 5 PRECEDING AND CURRENT ROW) AS ar,
   ARMA_PRED(b) OVER (ORDER BY rowtime ROWS  BETWEEN 5 PRECEDING AND CURRENT ROW) AS
arma,
   ARIMA_PRED(b) OVER (ORDER BY rowtime ROWS  BETWEEN 5 PRECEDING AND CURRENT ROW) AS
arima
FROM MyTable
```

Holt Winters

Holt Winters算法是Exponential smoothing方法中的一种，主要特点是可以捕捉时间序列中的**季节性**趋势。

语法格式

```
HOLT_WINTERS(field, seasonality, forecastOrder)
```

表 4-75 参数说明

参数	是否必选	说明
field	是	数据在数据流中的字段名。
seasonality	是	季节性变化的周期。例如数据点是按天采集，季节性周期是一周，则seasonality为7。
forecastOrder	否	指定需要预测的元素。 当forecastOrder为1，预测下一个元素。 当forecastOrder为2，预测下下一个元素。默认值为1。 使用此参数时需要保证over窗口的大小大于设置的forecastOrder。

示例

使用HOLT WINTERS函数结合窗口进行时间序列预测。

```
SELECT b,
   HOLT_WINTERS(b, 5) OVER (ORDER BY rowtime ROWS  BETWEEN 5 PRECEDING AND CURRENT ROW) AS a1,
   HOLT_WINTERS(b, 5, 2) OVER (ORDER BY rowtime ROWS  BETWEEN 5 PRECEDING AND CURRENT ROW) AS a2
FROM MyTable
```

4.1.18.3 实时聚类

聚类算法是非监督算法中非常典型的一类算法，经典的K-Means算法通过提前确定类别数目，计算数据点之间的距离来分类。对于离线静态数据集，我们可以依赖领域中知识来确定类别数目，运行K-Means算法可以取得比较好的聚类效果。但是对于在线实时流数据，数据是在不断变化和演进，类别数目极有可能发生变化，DLI服务提供一种能够应对此类场景，无需提前设定聚类数目，并且低延时的在线聚类算法。

算法大致思想为：定义一种距离函数，两两数据点之间如果距离小于某个阈值，则他们属于同一个类别。若某数据点和多个类别中心点的距离都小于这个阈值，则多个类别会发生合并操作。当数据流中的数据到达，算法会分别计算与所有类别的距离，从而决定此数据作为一个新类别或者归属于某类别。

语法格式

```
CENTROID(ARRAY[field_names], distance_threshold)：加入当前数据点后，该数据点所属分类中心。
CLUSTER_CENTROIDS(ARRAY[field_names], distance_threshold)：加入当前数据点后，所有分类中心。
ALL_POINTS_OF_CLUSTER(ARRAY[field_names], distance_threshold)：加入当前数据点后，该分类所有数据点。
ALL_CLUSTERS_POINTS(ARRAY[field_names], distance_threshold)：加入当前数据点后，所有分类对应的所有数据点。
```

说明

- 聚类算法可以应用在**无界流**中。

参数说明

表 4-76 参数说明

参数	是否必选	说明
field_names	是	数据在数据流中的字段名，多字段以逗号隔开。例如ARRAY[a, b, c]。
distance_threshold	是	距离阈值，当两数据点距离小于阈值时，它们将属于同一个类别。

示例

分别使用四种函数结合窗口来实时计算聚类的相关信息。

```
SELECT
  CENTROID(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE UNBOUNDED PRECEDING) AS centroid,
  CLUSTER_CENTROIDS(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE UNBOUNDED PRECEDING) AS
centroids
FROM MyTable

SELECT
  CENTROID(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE BETWEEN INTERVAL '60' MINUTE
PRECEDING AND CURRENT ROW) AS centroidCE,
  ALL_POINTS_OF_CLUSTER(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE BETWEEN INTERVAL '60'
MINUTE PRECEDING AND CURRENT ROW) AS itemList,
  ALL_CLUSTERS_POINTS(ARRAY[c,e], 1.0) OVER (ORDER BY proctime RANGE  BETWEEN INTERVAL '60'
MINUTE PRECEDING AND CURRENT ROW) AS listoflistofpoints
FROM MyTable
```

4.1.18.4 深度学习模型预测

深度学习已经广泛应用于图像分类、图像识别和语音识别等不同领域，DLI服务中提供了若干函数实现加载深度学习模型并进行预测的能力。

目前可支持的模型包括DeepLearning4j 模型和Keras模型。由于Keras它能够以TensorFlow、CNTK或者 Theano 作为后端运行，导入来自Keras的神经网络模型，可以借此导入Theano、Tensorflow、Caffe、CNTK等主流学习框架的模型。

语法格式

```
-- 图像分类， 返回预测图像分类的类别id
DL_IMAGE_MAX_PREDICTION_INDEX(field_name, model_path, is_dl4j_model)
DL_IMAGE_MAX_PREDICTION_INDEX(field_name, keras_model_config_path, keras_weights_path) -- 适用于
Keras模型

-- 文本分类，返回预测文本分类的类别id
DL_TEXT_MAX_PREDICTION_INDEX(field_name, model_path, is_dl4j_model) -- 采用默认word2vec模型
DL_TEXT_MAX_PREDICTION_INDEX(field_name, word2vec_path, model_path, is_dl4j_model)
```

说明

模型及配置文件等需存储在用户的OBS中，路径格式为"obs://**your_ak**:**your_sk**@obs.**your_obs_region**.xxx.com:443/**your_model_path**"。

参数说明

表 4-77 参数说明

参数	是否必选	说明
field_name	是	数据在数据流中的字段名。 图像分类中field_name类型需声明为ARRAY[TINYINT]。 文本分类中field_name类型需声明为String。
model_path	是	模型存放在OBS上的完整路径，包括模型结构和模型权值。
is_dl4j_model	是	是否是deeplearning4j的模型。 true代表是deeplearning4j，false代表是keras模型。
keras_model_config_path	是	模型结构存放在OBS上的完整路径。在keras中通过model.to_json()可得到模型结构。
keras_weights_path	是	模型权值存放在OBS上的完整路径。在keras中通过model.save_weights(filepath)可得到模型权值。
word2vec_path	是	word2vec模型存放在OBS上的完整路径。

示例

图片分类预测我们采用Mnist数据集作为流的输入，通过加载预训练的deeplearning4j模型或者keras模型，可以实时预测每张图片代表的数字。

```
CREATE SOURCE STREAM Mnist(
    image Array[TINYINT]
)
SELECT DL_IMAGE_MAX_PREDICTION_INDEX(image, 'your_dl4j_model_path', false) FROM Mnist
SELECT DL_IMAGE_MAX_PREDICTION_INDEX(image, 'your_keras_model_path', true) FROM Mnist
SELECT DL_IMAGE_MAX_PREDICTION_INDEX(image, 'your_keras_model_config_path', 'keras_weights_path')
FROM Mnist
```

文本分类预测我们采用一组新闻标题数据作为流的输入，通过加载预训练的deeplearning4j模型或者keras模型，可以实时预测每个新闻标题所属的类别，比如经济，体育，娱乐等。

```
CREATE SOURCE STREAM News(
    title String
)
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title, 'your_dl4j_word2vec_model_path','your_dl4j_model_path',
false) FROM News
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title,
'your_keras_word2vec_model_path','your_keras_model_path', true) FROM News
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title, 'your_dl4j_model_path', false) FROM New
SELECT DL_TEXT_MAX_PREDICTION_INDEX(title, 'your_keras_model_path', true) FROM New
```

4.1.19 保留关键字

Flink SQL将一些字符串组合保留为关键字以备将来使用。 如果要使用以下字符串作为字段名称，请确保使用反引号（例如`value`，`count`）。

A

- A
- ABS
- ABSOLUTE
- ACTION
- ADA
- ADD
- ADMIN
- AFTER
- AK
- ALL
- ALLOCATE
- ALLOW
- ALTER
- ALWAYS
- AND
- ANY
- APPEND
- APP_ID
- ARE
- ARRAY
- ARRAY_BRACKET
- AS
- ASC
- ASENSITIVE
- ASSERTION
- ASSIGNMENT
- ASYMMETRIC
- AT
- AT_LEAST_ONCE
- ATOMIC
- ATTRIBUTE
- ATTRIBUTES
- AUTHORIZATION

- AVG
- AVRO_CONFIG
- AVRO_DATA
- AVRO_SCHEMA

B

- BATCH_INSERT_DATA_NUM
- BEFORE
- BEGIN
- BERNOULLI
- BETWEEN
- BIGINT
- BINARY
- BIT
- BLOB
- BOOL
- BOOLEAN
- BOTH
- BREADTH
- BUCKET
- BY

C

- C
- CACHE_MAX_NUM
- CACHE_TIME
- CALL
- CALLED
- CARDINALITY
- CASCADE
- CASCADED
- CASE
- CAST
- CATALOG
- CATALOG_NAME
- CEIL
- CEILING
- CENTURY
- CHAIN

- CHANNEL
- CHAR
- CHARACTER
- CHARACTERISTICTS
- CHARACTERS
- CHARACTER_LENGTH
- CHARACTER_SET_CATALOG
- CHARACTER_SET_NAME
- CHARACTER_SET_SCHEMA
- CHAR_LENGTH
- CHECK
- CHECKPOINT_APP_NAME
- CHECKPOINT_INTERVAL
- CHECKPOINTINTERVAL
- CLASS_ORIGIN
- CLOB
- CLOSE
- CLUSTER_ADDRESS
- CLUSTER_ID
- CLUSTER_NAME
- COALESCE
- COBOL
- COLLATE
- COLLATION
- COLLATION_CATALOG
- COLLATION_NAME
- COLLATION_SCHEMA
- COLLECT
- COLUMN
- COLUMN_NAME
- COLUMN_NAME_MAP
- COMMAND_FUNCTION
- COMMAND_FUNCTION_CODE
- COMMIT
- COMMITTED
- CONDITION
- CONDITION_NUMBER
- CONFIGURATION

- CONFLUENT_CERTIFICATE_NAME
- CONFLUENT_PROPERTIES
- CONFLUENT_SCHEMA_FIELD
- CONFLUENT_URL
- CONNECT
- CONNECTION_NAME
- CONSTRAINT
- CONSTRAINTS
- CONSTRAINT_CATALOG
- CONSTRAINT_NAME
- CONSTRAINT_SCHEMA
- CONSTRUCTOR
- CONTAINS
- CONTINUE
- CONVERT
- CORR
- CORRESPONDING
- COUNT
- COVAR_POP
- COVAR_SAMP
- CREATE
- CREATE_IF_NOT_EXIST
- CROSS
- CUBE
- CUME_DIST
- CURRENT
- CURRENT_CATALOG
- CURRENT_DATE
- CURRENT_DEFAULT_TRANSFORM_GROUP
- CURRENT_PATH
- CURRENT_ROLE
- CURRENT_SCHEMA
- CURRENT_TIMESTAMP
- CURRENT_TRANSFORM_GROUP_FOR_TYPE
- CURRENT_USER
- CURSOR
- CURSOR_NAME
- CYCLE

D

- DATE
- DATABASE
- DATE
- DATETIME_INTERVAL_CODE
- DATETIME_INTERVAL_PRECISION
- DAY
- DB_COLUMNS
- DB_URL
- DB_OBS_SERVER
- DB_TYPE
- DEALLOCATE
- DEC
- DECADE
- DECIMAL
- DECLARE
- DEFAULTS
- DEFERRABLE
- DEFERRED
- DEFINER
- DEGREE
- DELETE
- DELETE_OBS_TEMP_FILE
- DENSE_RANK
- DEPTH
- DEREF
- DERIVED
- DESC
- DESCRIBE
- DESCRIPTION
- DESCRIPTOR
- DETERMINISTIC
- DIAGNOSTICS
- DISALLOW
- DISCONNECT
- DIS_NOTICE_CHANNEL
- DISPATCH
- DISTINCT

- DOMAIN
- DOUBLE
- DOW
- DOY
- DRIVER
- DROP
- DUMP_INTERVAL
- DYNAMIC
- DYNAMIC_FUNCTION
- DYNAMIC_FUNCTION_CODE

E

- EACH
- ELEMENT
- ELSE
- EMAIL_KEY
- ENABLECHECKPOINT
- ENABLE_CHECKPOINT
- ENABLE_OUTPUT_NULL
- ENCODE
- ENCODE_CLASS_NAME
- ENCODE_CLASS_PARAMETER
- ENCODED_DATA
- END
- ENDPOINT
- END_EXEC
- EPOCH
- EQUALS
- ESCAPE
- ES_FIELDS
- ES_INDEX
- ES_TYPE
- ESTIMATEMEM
- ESTIMATEPARALLELISM
- EXACTLY_ONCE
- EXCEPT
- EXCEPTION
- EXCLUDE
- EXCLUDING

- EXEC
- EXECUTE
- EXISTS
- EXP
- EXPLAIN
- EXTEND
- EXTERNAL
- EXTRACT
- EVERY

F

- FALSE
- FETCH
- FIELD_DELIMITER
- FIELD_NAMES
- FILE_PREFIX
- FILTER
- FINAL
- FIRST
- FIRST_VALUE
- FLOAT
- FLOOR
- FOLLOWING
- FOR
- FUNCTION
- FOREIGN
- FORTRAN
- FOUND
- FRAC_SECOND
- FREE
- FROM
- FULL
- FUSION

G

- G
- GENERAL
- GENERATED
- GET

- GLOBAL
- GO
- GOTO
- GRANT
- GRANTED
- GROUP
- GROUPING
- GW_URL

H

- HASH_KEY_COLUMN
- HAVING
- HIERARCHY
- HOLD
- HOUR
- HTTPS_PORT

I

- IDENTITY
- ILLEGAL_DATA_TABLE
- IMMEDIATE
- IMPLEMENTATION
- IMPORT
- IN
- INCLUDING
- INCREMENT
- INDICATOR
- INITIALLY
- INNER
- INOUT
- INPUT
- INSENSITIVE
- INSERT
- INSTANCE
- INSTANTIABLE
- INT
- INTEGER
- INTERSECT
- INTERSECTION

- INTERVAL
- INTO
- INVOKER
- IN_WITH_SCHEMA
- IS
- ISOLATION

J

- JAVA
- JOIN
- JSON_CONFIG
- JSON_SCHEMA

K

- K
- KAFKA_BOOTSTRAP_SERVERS
- KAFKA_CERTIFICATE_NAME
- KAFKA_GROUP_ID
- KAFKA_PROPERTIES
- KAFKA_PROPERTIES_DELIMITER
- KAFKA_TOPIC
- KEY
- KEY_COLUMN
- KEY_MEMBER
- KEY_TYPE
- KEY_VALUE
- KRB_AUTH

L

- LABEL
- LANGUAGE
- LARGE
- LAST
- LAST_VALUE
- LATERAL
- LEADING
- LEFT
- LENGTH
- LEVEL

- LIBRARY
- LIKE
- LIMIT
- LONG

M

- M
- MAP
- MATCH
- MATCHED
- MATCHING_COLUMNS
- MATCHING_REGEX
- MAX
- MAXALLOWEDCPU
- MAXALLOWEDMEM
- MAXALLOWEDPARALLELISM
- MAX_DUMP_FILE_NUM
- MAX_RECORD_NUM_CACHE
- MAX_RECORD_NUM_PER_FILE
- MAXVALUE
- MEMBER
- MERGE
- MESSAGE_COLUMN
- MESSAGE_LENGTH
- MESSAGE_OCTET_LENGTH
- MESSAGE_SUBJECT
- MESSAGE_TEXT
- METHOD
- MICROSECOND
- MILLENNIUM
- MIN
- MINUTE
- MINVALUE
- MOD
- MODIFIES
- MODULE
- MONTH
- MORE
- MS

- MULTISET
- MUMPS

N

- NAME
- NAMES
- NATIONAL
- NATURAL
- NCHAR
- NCLOB
- NESTING
- NEW
- NEXT
- NO
- NONE
- NORMALIZE
- NORMALIZED
- NOT
- NULL
- NULLABLE
- NULLIF
- NULLS
- NUMBER
- NUMERIC

O

- OBJECT
- OBJECT_NAME
- OBS_DIR
- OCTETS
- OCTET_LENGTH
- OF
- OFFSET
- OLD
- ON
- ONLY
- OPEN
- OPERATION_FIELD
- OPTION

- OPTIONS
- OR
- ORDER
- ORDERING
- ORDINALITY
- OTHERS
- OUT
- OUTER
- OUTPUT
- OVER
- OVERLAPS
- OVERLAY
- OVERRIDING

P

- PAD
- PARALLELISM
- PARAMETER
- PARAMETER_MODE
- PARAMETER_NAME
- PARAMETER_ORDINAL_POSITION
- PARAMETER_SPECIFIC_CATALOG
- PARAMETER_SPECIFIC_NAME
- PARAMETER_SPECIFIC_SCHEMA
- PARTIAL
- PARTITION
- PARTITION_COUNT
- PARTITION_KEY
- PARTITION_RANGE
- PASCAL
- PASSTHROUGH
- PASSWORD
- PATH
- PERCENTILE_CONT
- PERCENTILE_DISC
- PERCENT_RANK
- PERSIST_SCHEMA
- PIPELINE_ID
- PLACING

- PLAN
- PLI
- POSITION
- POWER
- PRECEDING
- PRECISION
- PREPARE
- PRESERVE
- PRIMARY
- PRIMARY_KEY
- PRIOR
- PRIVILEGES
- PROCEDURE
- PROCTIME
- PROJECT_ID
- PUBLIC

Q

- QUARTER
- QUOTE

R

- RANGE
- RANK
- RAW
- READ
- READS
- READ_ONCE
- REAL
- RECURSIVE
- REF
- REFERENCES
- REFERENCING
- REGION
- 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
- ROLLING_INTERVAL
- ROLLING_SIZE
- ROLLUP
- ROUTINE
- ROUTINE_CATALOG
- ROUTINE_NAME
- ROUTINE_SCHEMA
- ROW
- ROW_COUNT
- ROW_DELIMITER
- ROW_NUMBER
- ROWS
- ROWTIME

S

- SAVEPOINT
- SCALE
- SCHEMA

- SCHEMA_CASE_SENSITIVE
- 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
- SINK
- SIZE
- SK
- 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
- START_TIME
- STATE
- STATEMENT
- STATIC
- STDDEV_POP
- STDDEV_SAMP
- STREAM
- STRING
- STRUCTURE
- STYLE
- SUBCLASS_ORIGIN
- SUBMULTISET
- SUBSTITUTE
- SUBSTRING
- SUM
- SYMMETRIC
- SYSTEM
- SYSTEM_USER

T

- TABLE
- TABLESAMPLE
- TABLE_COLUMNS
- TABLE_NAME
- TABLE_NAME_MAP
- TEMP
- TEMPORARY

- THEN
- TIES
- TIME
- TIMESTAMP
- TIMESTAMPADD
- TIMESTAMPDIFF
- TIMEZONE_HOUR
- TIMEZONE_MINUTE
- TINYINT
- TO
- TOP_LEVEL_COUNT
- TOPIC
- TOPIC_URN
- TRAILING
- TRANSACTION
- TRANSACTIONAL_TABLE
- TRANSACTIONS_ACTIVE
- TRANSACTIONS_COMMITTED
- TRANSACTIONS_ROLLED_BACK
- TRANSFORM
- TRANSFORMS
- TRANSLATE
- TRANSLATION
- TRANX_ID
- TREAT
- TRIGGER
- TRIGGER_CATALOG
- TRIGGER_NAME
- TRIGGER_SCHEMA
- TRIM
- TRUE
- TSDB_LINK_ADDRESS
- TSDB_METRICS
- TSDB_TIMESTAMPS
- TSDB_TAGS
- TSDB_VALUES
- TYPE
- TYPE_CLASS_NAME

- TYPE_CLASS_PARAMETER

U

- UESCAPE
- UNBOUNDED
- UNCOMMITTED
- UNDER
- UNION
- UNIQUE
- UNKNOWN
- UNNAMED
- UNNEST
- UPDATE
- UPPER
- UPSERT
- URN_COLUMN
- USAGE
- USER
- USER_DEFINED_TYPE_CATALOG
- USER_DEFINED_TYPE_CODE
- USER_DEFINED_TYPE_NAME
- USER_DEFINED_TYPE_SCHEMA
- USERNAME
- USING

V

- VALUE
- VALUES
- VALUE_TYPE
- VARBINARY
- VARCHAR
- VARYING
- VAR_POP
- VAR_SAMP
- VERSION
- VERSION_ID
- VIEW

W

- WATERMARK
- WEEK
- WHEN
- WHENEVER
- WHERE
- WIDTH_BUCKET
- WINDOW
- WITH
- WITHIN
- WITHOUT
- WORK
- WRAPPER
- WRITE

X

- XML
- XML_CONFIG

Y

- YEAR

Z

- ZONE

5 标示符

5.1 aggregate_func

格式

无。

说明

聚合函数。

5.2 alias

格式

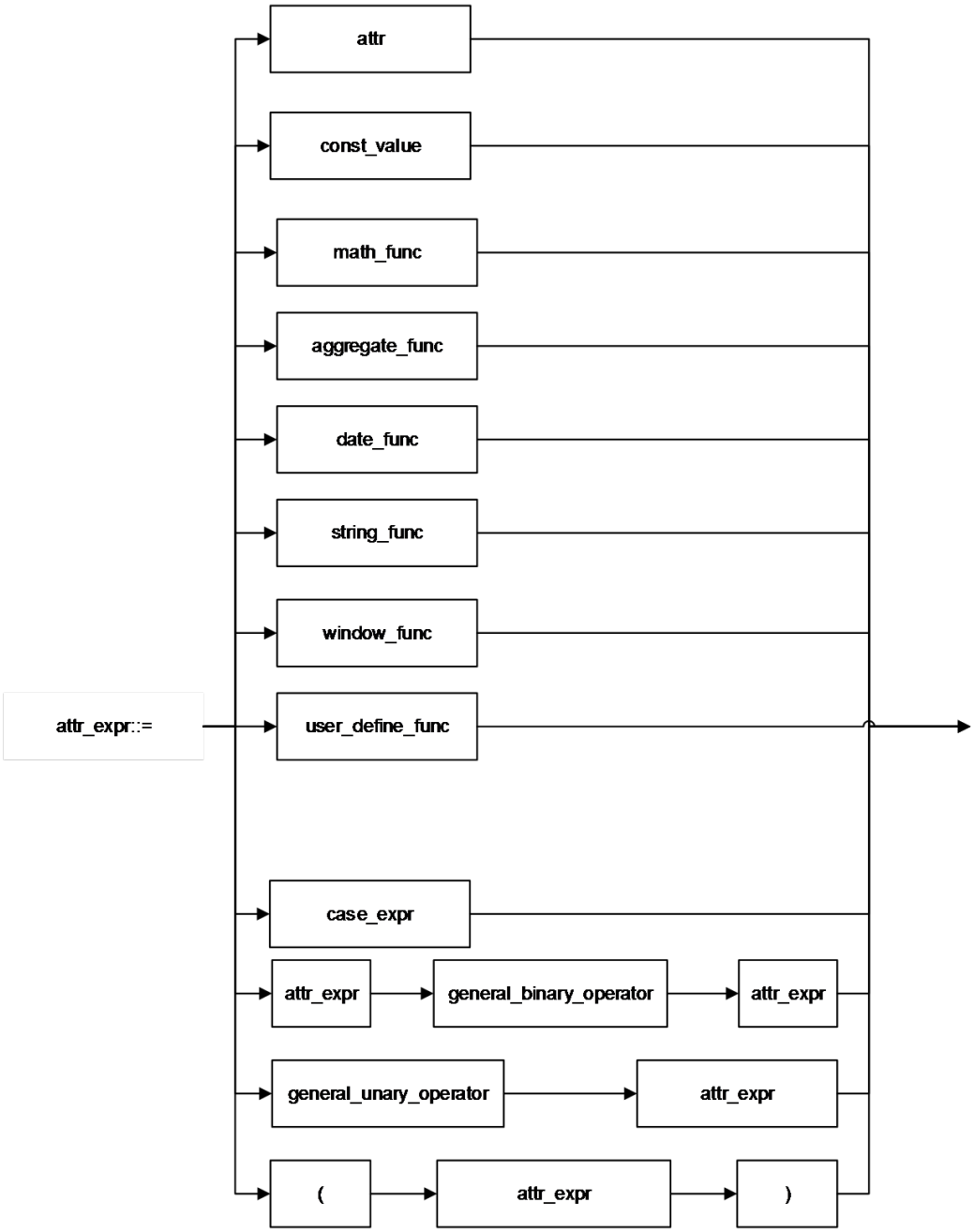
无。

说明

别名，可给字段、表、视图、子查询起别名，仅支持字符串类型。

5.3 attr_expr

格式



说明

语法	描述
attr_expr	属性表达式。

语法	描述
attr	表的字段，与col_name相同。
const_value	常量值。
case_expr	case表达式。
math_func	数学函数。
date_func	日期函数。
string_func	字符串函数。
aggregate_func	聚合函数。
window_func	分析窗口函数。
user_define_func	用户自定义函数。
general_binary_operator	普通二元操作符。
general_unary_operator	普通一元操作符。
(	指定子属性表达式开始。
)	指定子属性表达式结束。

5.4 attr_expr_list

格式

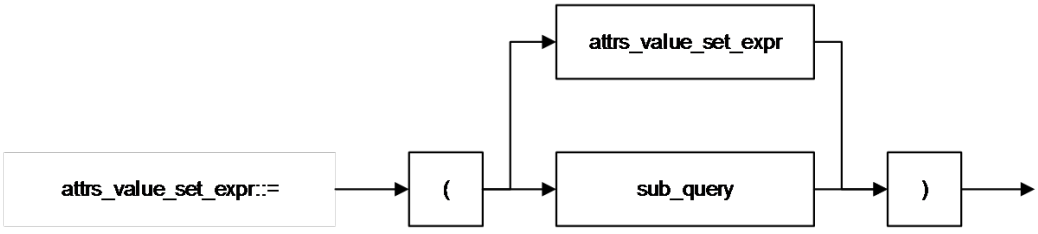
无。

说明

attr_expr列表，以逗号分隔。

5.5 attrs_value_set_expr

格式



说明

语法	描述
attrs_value_set_expr	属性值集合。
sub_query	子查询语句。
(	指定子查询表达式开始。
)	指定子查询表达式结束。

5.6 boolean_expression

格式

无。

说明

返回boolean类型的表达式。

5.7 col

格式

无。

说明

函数调用时的形参，一般即为字段名称，与col_name相同。

5.8 col_comment

格式

无。

说明

对列（字段）的描述，仅支持字符串类型，描述长度不能超过256字节。

5.9 col_name

格式

无。

说明

列名，即字段名称，仅支持字符串类型，名称长度不能超过128个字节。

5.10 col_name_list

格式

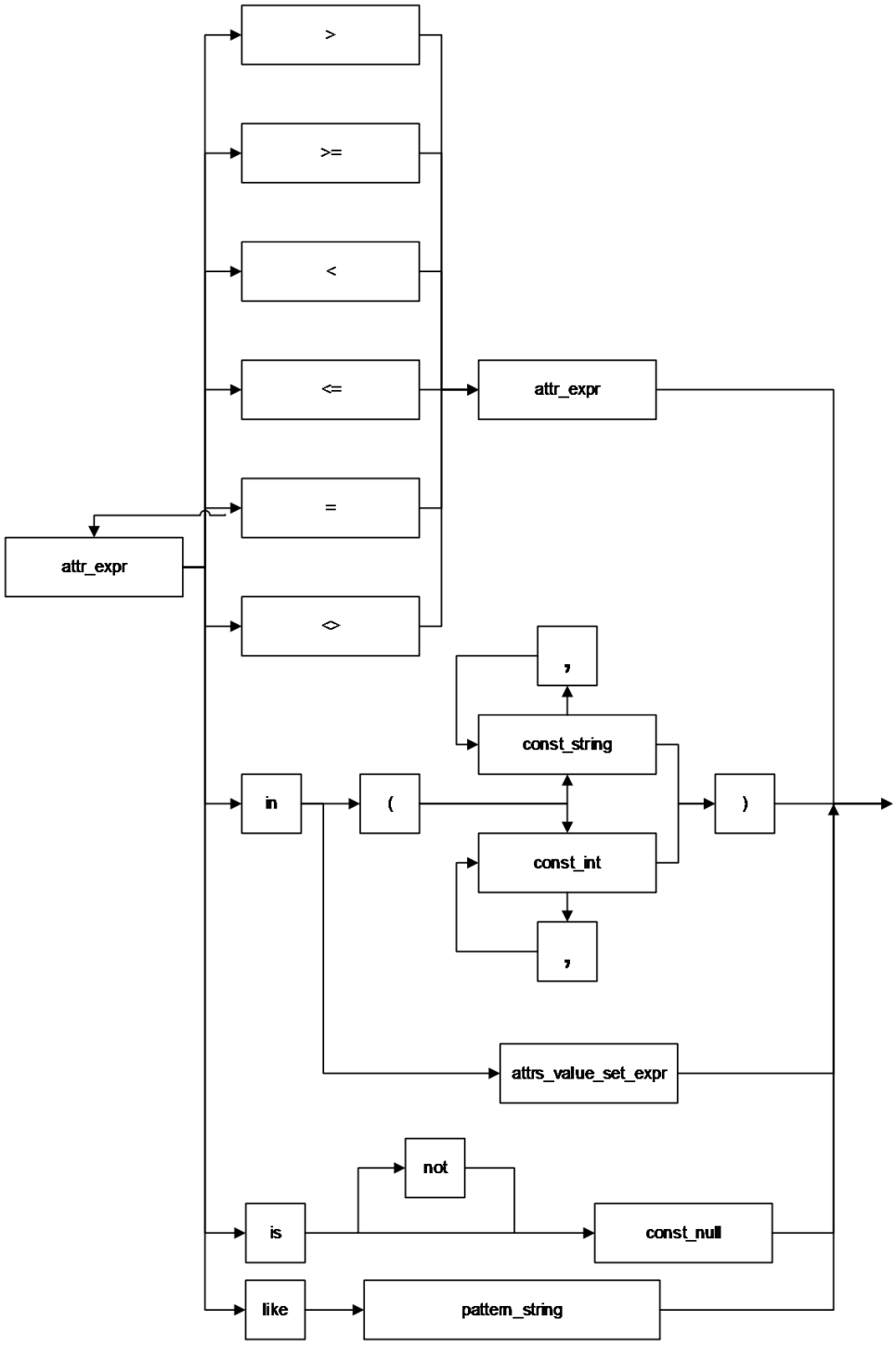
无。

说明

字段列表，可由一个或多个col_name构成，多个col_name之间用逗号分隔。

5.11 condition

格式

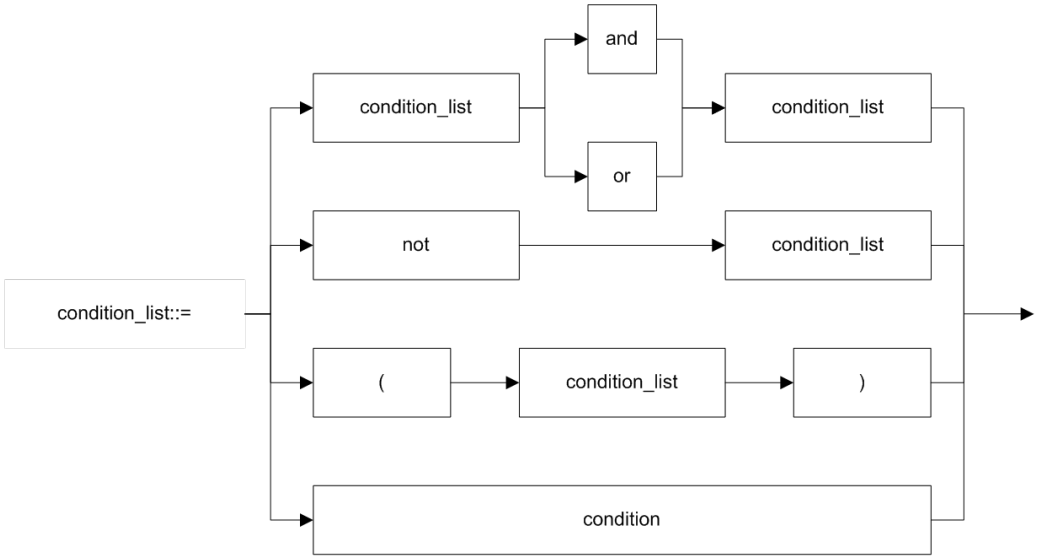


说明

语法	描述
condition	逻辑判断条件。
>	关系运算符：大于。
>=	关系运算符：大于等于。
<	关系运算符：小于。
<=	关系运算符：小于等于。
=	关系运算符：等于。
<>	关系运算符：不等于。
is	关系运算符：是。
is not	关系运算符：不是。
const_null	常量：空值。
like	关系运算符：用于通配符匹配。
pattern_string	模式匹配字符串，支持通配符匹配。WHERE LIKE条件过滤时，支持SQL通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。
attr_expr	属性表达式。
attrs_value_set_expr	属性值集合。
in	关键字，用于判断属性是否在一个集合中。
const_string	字符串常量。
const_int	整型常量。
(	指定常量集合开始。
)	指定常量集合结束。
,	逗号分隔符。

5.12 condition_list

格式



说明

语法	描述
condition_list	逻辑判断条件列表。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。

5.13 cte_name

格式

无。

说明

公共表达式的名字。

5.14 data_type

格式

无。

说明

数据类型，当前只支持原生数据类型。

5.15 db_comment

格式

无。

说明

对数据库的描述，仅支持字符串类型，描述长度不能超过256字节。

5.16 db_name

格式

无。

说明

数据库名称，仅支持字符串类型，名称长度不能超过128字节。

5.17 else_result_expression

格式

无。

说明

CASE WHEN语句中ELSE语句后的返回结果。

5.18 file_format

格式

| AVRO

| CSV

| JSON

| ORC

| PARQUET

说明

- 目前包含以上6种格式。
- 指定数据格式的方式有两种，一种是USING，可指定以上6种数据格式，另一种是STORED AS，只能指定ORC和PARQUET。
- ORC对RCFile做了优化，可以提供一种高效的方法来存储Hive数据。
- PARQUET是面向分析型业务的列式存储格式。

5.19 file_path

格式

无。

说明

文件路径，该路径是OBS路径。

5.20 function_name

格式

无。

说明

函数名称，仅支持字符串类型。

5.21 groupby_expression

格式

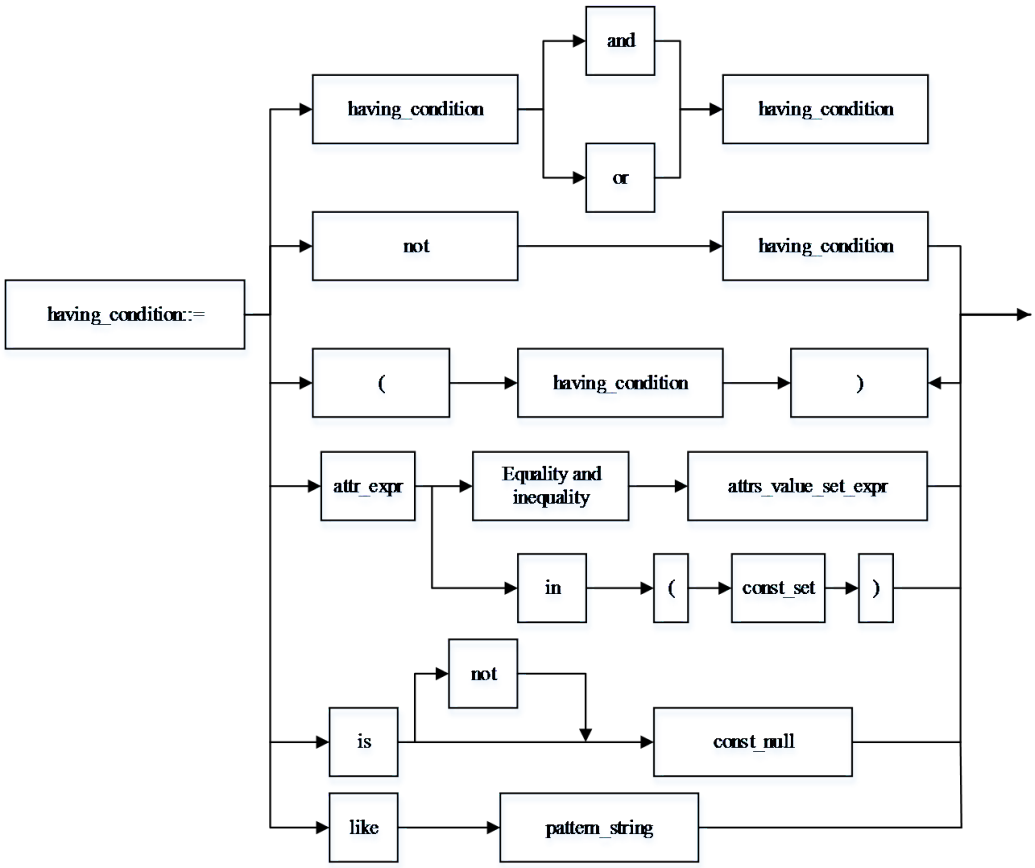
无。

说明

包含GROUP BY的表达式。

5.22 having_condition

格式



说明

语法	描述
having_condition	having逻辑判断条件。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。
const_set	常量集合，元素间逗号分隔。
in	关键字，用于判断属性是否在一个集合中。

语法	描述
attrs_value_set_expr	属性值集合。
attr_expr	属性表达式。
Equality and inequality	等式与不等式，详情请参见**关系运算符**。
pattern_string	模式匹配字符串，支持通配符匹配。WHERE LIKE条件过滤时，支持SQL通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。
like	关系运算符：用于通配符匹配。

5.23 input_expression

格式

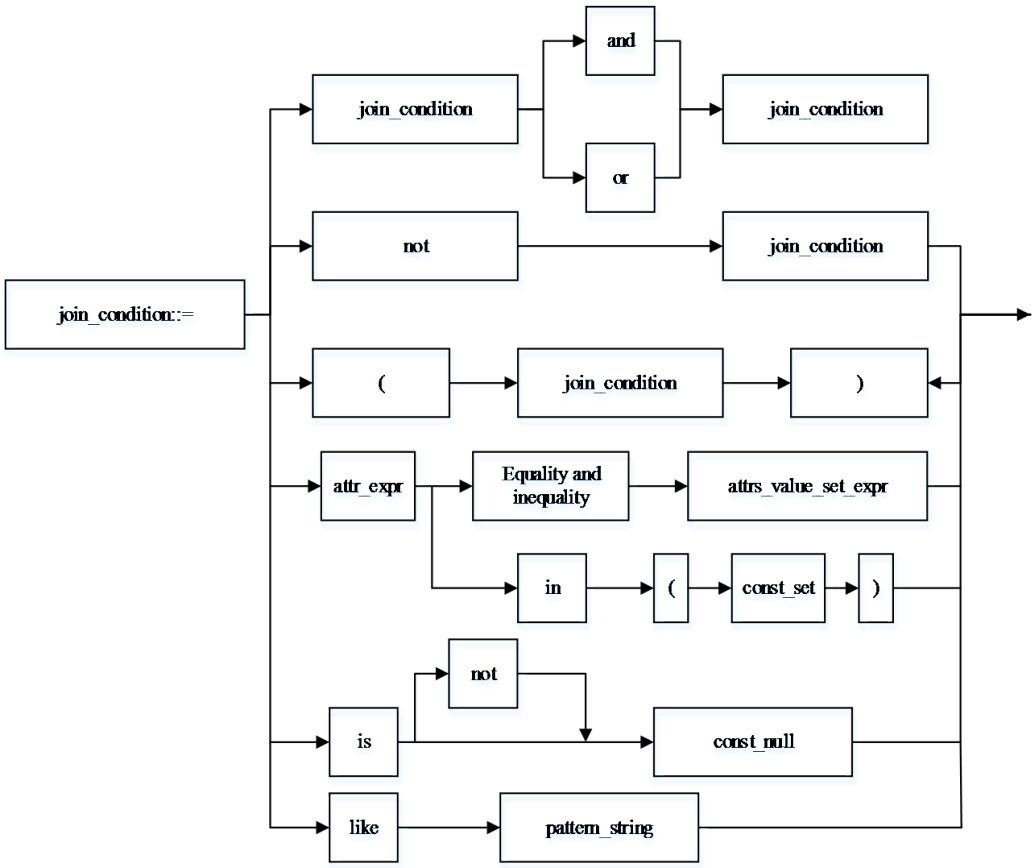
无。

说明

CASE WHEN的输入表达式。

5.24 join_condition

格式



说明

语法	描述
join_condition	join逻辑判断条件。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。
const_set	常量集合，元素间逗号分隔。
in	关键字，用于判断属性是否在一个集合中。

语法	描述
atrrs_value_set_expr	属性值集合。
attr_expr	属性表达式。
Equality and inequality	等式与不等式，详情请参见**关系运算符**。
pattern_string	模式匹配字符串，支持通配符匹配。WHERE LIKE条件过滤时，支持SQL通配符中“%”与“_”，“%”代表一个或多个字符，“_”仅代表一个字符。

5.25 non_equi_join_condition

格式

无。

说明

指不等式join条件。

5.26 number

格式

无。

说明

LIMIT限制输出的行数，只支持INT类型。

5.27 partition_col_name

格式

无。

说明

分区列名，即分区字段名称，仅支持字符串类型。

5.28 partition_col_value

格式

无。

说明

分区列值，即分区字段的值。

5.29 partition_specs

格式

partition_specs : (partition_col_name = partition_col_value, partition_col_name = partition_col_value, ...);

说明

表的分区列表，以key=value的形式表现，key为partition_col_name ，value为partition_col_value ，若存在多个分区字段，每组key=value之间用逗号分隔。

5.30 property_name

格式

无。

说明

属性名称，仅支持字符串类型。

5.31 property_value

格式

无。

说明

属性值，仅支持字符串类型。

5.32 regex_expression

格式

无。

说明

模式匹配字符串，支持通配符匹配。

5.33 result_expression

格式

无。

说明

CASE WHEN语句中THEN语句后的返回结果。

5.34 select_statement

格式

无。

说明

SELECT基本语句，即查询语句。

5.35 separator

格式

无。

说明

分隔符，仅支持CHAR类型，支持用户自定义，如逗号、分号、冒号等。

5.36 sql_containing_cte_name

格式

无。

说明

包含了cte_name定义的公共表达式的SQL语句。

5.37 sub_query

格式

无。

说明

指子查询。

5.38 table_comment

格式

无。

说明

对表的描述，仅支持字符串类型，描述长度不能超过256字节。

5.39 table_name

格式

无。

说明

表名称，支持字符串类型和“$”符号，名称长度不能超过128字节。

5.40 table_properties

格式

无。

说明

表的属性列表，以key=value的形式表示，key为property_name，value为property_value，列表中每组key=value之间用逗号分隔。

5.41 table_reference

格式

无。

说明

表或视图的名称，仅支持字符串类型，也可为子查询，当为子查询时，必须加别名。

5.42 when_expression

格式

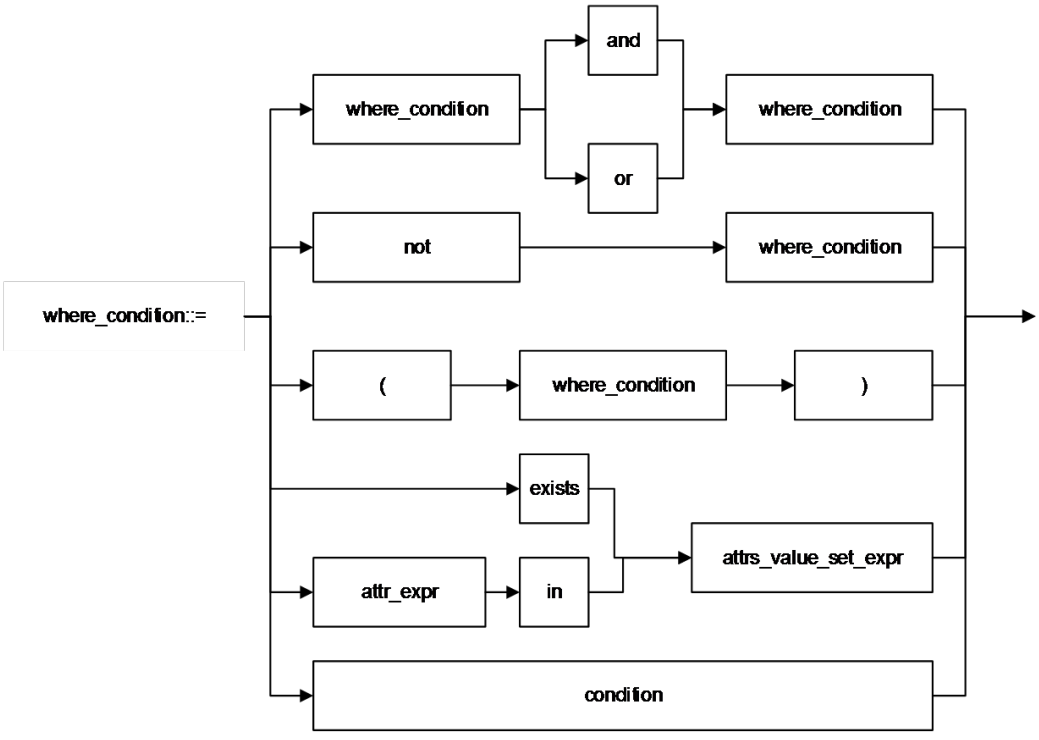
无。

说明

CASE WHEN语句的when表达式，与输入表达式进行匹配。

5.43 where_condition

格式



说明

语法	描述
where_condition	where逻辑判断条件。
and	逻辑运算符：与。
or	逻辑运算符：或。
not	逻辑运算符：非。
(	子逻辑判断条件开始。
)	子逻辑判断条件结束。
condition	逻辑判断条件。
exists	关键字，用于判断是否存在一个不为空的集合，若exists后面跟的为子查询，子查询中须包含逻辑判断条件。
in	关键字，用于判断属性是否在一个集合中。
attrs_value_set_expr	属性值集合。
attr_expr	属性表达式。

5.44 window_function

格式

无。

说明

分析窗口函数。

6 运算符

6.1 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。

DLI提供的关系运算符，请参见表6-1。

表 6-1 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相等，返回TRUE，否则返回FALSE。不能用于赋值操作。
A <=> B	BOOLEAN	若A与B相等，返回TRUE，否则返FALSE，若A与B都为NULL则返回TRUE，A与B其中一个为NULL则返回FALSE。
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	与<>逻辑操作符相同，该种运算符为SQL Server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。

运算符	返回类型	描述
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 BETWEEN B AND C	BOOLEAN	若A大于等于B且小于等于C则返回TRUE，否则返回FALSE。若A、B、C三者中存在NULL，则返回NULL。
A NOT BETWEEN B AND C	BOOLEAN	若A小于B或大于C则返回TRUE，否则返回FALSE。若A、B、C三者中存在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 LIKE B	BOOLEAN	若字符串A与字符串B相匹配则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A NOT LIKE B	BOOLEAN	若字符串A与字符串B不相匹配则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A RLIKE B	BOOLEAN	JAVA的LIKE操作，若A或其子字符串与B相匹配，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A REGEXP B	BOOLEAN	与A RLIKE B结果相同。

6.2 算术运算符

算术运算符包括双目运算与单目运算，这些运算符都将返回数字类型。DLI所支持的算术运算符如**表6-2**所示。

表 6-2 算术运算符

运算符	返回类型	描述
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。
A % B	所有数字类型	A对B取余数，结果数据之类与操作数据类型相关。

运算符	返回类型	描述
A & B	所有数字类型	查看两个参数的二进制表示法的值，并执行按位”与”操作。两个表达式的一位均为1时，则结果的该位为1。否则，结果的该位为0。
A | B	所有数字类型	查看两个参数的二进制表示法的值，并执行按位”或”操作。只要任一表达式的一位为1，则结果的该位为 1。否则，结果的该位为0。
A ^ B	所有数字类型	查看两个参数的二进制表示法的值，并执行按位”异或”操作。当且仅当只有一个表达式的某位上为1时，结果的该位才为1。否则结果的该位为0。
~A	所有数字类型	对一个表达式执行按位”非”操作（取反）。

6.3 逻辑运算符

常用的逻辑操作符有AND、OR和NOT，它们的运算结果有三个值，分别为TRUE、FALSE和NULL，其中NULL代表未知。优先级顺序为：NOT>AND>OR。

运算规则请参见表6-3，表中的A和B代表逻辑表达式。

表 6-3 逻辑运算符

运算符	返回类型	描述
A AND B	BOOLEAN	若A与B都为TRUE则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。
A OR B	BOOLEAN	若A或B为TRUE，则返回TRUE，否则返回FALSE。若A或B为NULL，则返回NULL。一个为TRUE，另一个为NULL时，返回TRUE。
NOT A	BOOLEAN	若A为FALSE则返回TRUE，若A为NULL则返回NULL，否则返回FALSE。
! A	BOOLEAN	与NOT A相同。
A IN (val1, val2, ...)	BOOLEAN	若A与(val1, val2, ...)中任意值相等则返回TRUE，否则返回FALSE。
A NOT IN (val1, val2, ...)	BOOLEAN	若A与(val1, val2, ...)中任意值都不相等则返回TRUE，否则返回FALSE。
EXISTS (subquery)	BOOLEAN	若子查询返回结果至少包含一行则返回TRUE，否则返回FALSE。

运算符	返回类型	描述
NOT EXISTS (subquery)	BOOLEAN	若子查询返回结果一行都不包含则返回TRUE，否则返回FALSE。