智能数据湖 AI DataLake

Flink 语法参考

文档版本	01
发布日期	2026-11-06



华为云计算技术有限公司

版权所有 © 华为云计算技术有限公司 2026。 保留一切权利。

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

商标声明


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

注意

您购买的产品、服务或特性等应受华为云计算技术有限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，华为云计算技术有限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。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 连接器.....1
1.1 连接器的支持情况.....1
1.2 Kafka 连接器.....2
1.3 Doris 连接器.....11
1.4 Upsert-Kafka 连接器.....19
1.5 FileSystem 连接器.....25
1.6 DataGen 连接器.....32
1.7 BlackHole 连接器.....39
1.8 Print 连接器.....42
2 关键字.....46
3 DDL 语句.....47
3.1 CREATE CATALOG.....47
3.2 CREATE DATABASE.....48
3.3 CREATE TABLE.....49
3.4 CREATE VIEW.....53
3.5 CREATE FUNCTION.....54
3.6 ALTER 语句.....56
3.7 DROP 语句.....58
4 DML 语句.....60
4.1 INSERT INTO.....60
5 Queries 语句.....63
6 内置函数.....65
6.1 比较函数.....65
6.2 逻辑函数.....68
6.3 算术函数.....70
6.4 字符串函数.....75
6.5 时间函数.....81
6.6 条件函数.....87
6.7 类型转换函数.....89
6.8 聚合函数.....91
6.9 JSON 函数.....95

 版权所有 © 华为云计算技术有限公司

6.10 值构造函数 99
6.11 值访问函数 100
6.12 分组函数 101
6.13 哈希函数 102
7 辅助语句 105
7.1 USE 语句 105
7.2 SET 语句 106
8 Flink 增强特性使用介绍 109
8.1 增强函数说明 109
8.1.1 BITAND 109
8.1.2 BITNOT 110
8.1.3 BITOR 110
8.1.4 BITXOR 111
8.1.5 DATE_ADD 112
8.1.6 DATE_SUB 113
8.1.7 DATEDIFF 113
8.1.8 URL_DECODE 114
8.1.9 URL_ENCODE 115
8.2 FlinkSQL 使用 RocksDB Merge Operator 提升 Join 算子性能 116
8.3 FlinkSQL 使用 RocksDB Prefix Bloom Filter 提升 Join 算子和 TopN 算子性能 117
8.4 FlinkSQL SPLIT_INDEX 函数性能优化 117
8.5 FlinkSQL 使用 Object 复用特性提升 FlinkSQL 计算性能 118
8.6 FlinkSQL Lookup Join 支持 PROCTIME()语法 119
8.7 FlinkSQL JSON_VALUE 函数性能优 120
8.8 FlinkSQL Kafka 和 upsert-kafka Connector 支持限流读 120
8.9 FlinkSQL 支持 Multi Join 121
8.10 Flink On Iceberg 支持维表 Join 123
8.11 Flink On Iceberg 支持 VARIANT 数据类型 124

 版权所有 © 华为云计算技术有限公司

1 连接器

1.1 连接器的支持情况

连接器的支持情况请参考**表1-1**。

Flink支持Iceberg、Paimon两类连接器。关于这两种连接器的完整使用说明（含使用前准备、数据准备等详细内容），已统一整理在**Iceberg on Flink**和**Paimon on Flink**章节；**表1-1**仅作能力覆盖说明。

表 1-1 连接器的支持情况

连接器	源表	维表	结果表	说明	组件版本
1.2 Kafka连接器	√	×	√	读写DMS Kafka消息队列数据。	DMS Kafka需要为3.x及以上版本。
Iceberg连接器	√	√	√	读写Iceberg湖仓表数据。	不涉及
Paimon连接器	√	√	√	读写Paimon湖仓表数据（流批一体）。	不涉及
1.3 Doris连接器	√	√	√	读写CloudTable Doris数据库数据。	CloudTable Doris需要为3.x及以上版本。
1.4 Upsert-Kafka连接器	√	×	√	以Upsert语义读写DMS Kafka数据。	DMS Kafka需要为3.x及以上版本。
1.5 FileSystem连接器	√	×	√	读写本地文件系统、OBS等存储数据，支持批流统一读写。	不涉及

版权所有 © 华为云计算技术有限公司

连接器	源表	维表	结果表	说明	组件版本
1.6 DataGen连接器	√	×	×	按规则生成随机或序列测试数据，无需外部数据源。	不涉及
1.7 BlackHole连接器	×	×	√	丢弃所有写入的数据，不产生输出。	不涉及
1.8 Print连接器	×	×	√	Print连接器：将数据打印到标准输出，用于开发调试。	不涉及

1.2 Kafka 连接器

概述

Kafka连接器用于从**DMS Kafka主题（Topic）**读取数据或将数据写入DMS Kafka主题，是最常用的流式数据连接器，支持精确一次（Exactly-Once）语义保证。

表 1-2 连接器详情

类别	详情
支持类型	源表和结果表，数据摄入目标端
运行模式	流模式
数据格式	详情请参见**Formats**。
API种类	SQL，Datastream
是否支持更新或删除结果表数据	不支持更新和删除结果表数据，只支持插入数据。 更新和删除数据相关功能请参见**1.4 Upsert-Kafka连接器**。

适用场景

- **实时数据采集**：从Kafka消费消息，进行流式处理和分析。
- **数据管道**：将处理后的结果数据写入Kafka，供下游系统消费。
- **事件驱动架构**：基于Kafka消息触发实时计算和响应。
- **数据集成**：作为不同系统间的消息中间件，实现数据流转。
- **CDC数据捕获**：配合Debezium/Canal格式，实现变更数据捕获。

约束与限制

- 消费者偏移量提交行为：开启Checkpoint时在Checkpoint完成时提交；未开启时依赖Kafka Consumer自动提交。

版权所有 © 华为云计算技术有限公司

- 默认消费数据时，topic须提前创建。
- DMS Kafka需要为3.x及以上版本。

语法结构

```
CREATE TABLE {table_name} (
  {column_name} {column_type},
  ...
) WITH (
  'connector' = 'kafka',
  'topic' = '{kafka_topic}',
  'properties.bootstrap.servers' = '{broker_list}',
  'properties.group.id' = '{consumer_group}',
  'scan.startup.mode' = '{startup_mode}',
  'format' = '{data_format}',
  '{format_specific_param}' = '{value}'
);
```

WITH 参数说明

Source参数

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	kafka	标识连接器类型为Kafka连接器。 必须填写且值固定为kafka。Flink运行时根据此值加载对应的Kafka Source/Sink工厂类，将表声明为Kafka数据源或输出表。
topic	是	String	无	有效的Topic名	指定消费的Kafka主题名称。 Source必填，支持分号分隔的多个Topic（如topic-a;topic-b）。Flink运行时为每个Topic分区分配一个子任务并行消费，多个Topic的分区将被合并调度。
properties.bootstrap.servers	是	String	无	host:port列表	指定Kafka Broker地址列表，多个地址以逗号分隔。 Source和Sink均必填。Flink运行时使用此配置创建Kafka Consumer或Producer客户端连接，建议配置多个Broker地址以实现高可用。
properties.group.id	否	String	无	有效的消费者组名	指定Kafka消费者组ID。 Source必填。Flink运行时使用此组ID标识消费者身份，相同组ID的消费者共享Topic分区的消费进度，不同作业应使用不同的group.id以避免分区分配冲突。

 版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
scan.startup.mode	否	String	group-offsets	earliest-offset / latest-offset / group-offsets / specific-offsets / timestamp	指定Source首次消费时的起始位置模式。 默认为group-offsets。 • 当设置为specific-offsets时需配合scan.startup.specific-offsets使用。 • 当设置为timestamp时需配合scan.startup.timestamp-millis使用。 Flink运行时根据此模式确定Kafka Consumer的初始消费位移，影响作业启动时的数据读取起点。
scan.startup.specific-offsets	否	String	无	partition:offset列表	指定各分区的起始偏移量。 仅在scan.startup.mode为specific-offsets时生效。格式为partition:offset,partition:offset。Flink运行时将Kafka Consumer的各分区初始位移设置到指定位置，从未消费过的分区将从该偏移量开始读取。
scan.startup.timestamp-millis	否	Long	无	毫秒时间戳	指定消费起始时间戳（毫秒）。 仅在scan.startup.mode为timestamp时生效。Flink运行时将Kafka Consumer的起始位移定位到不早于该时间戳的第一条消息，适用于从特定时间点恢复数据消费。
format	是	String	无	json / csv / avro / debezium-json / canal-json / raw	指定消息的反序列化格式。 Source和Sink均必填。Flink运行时根据此值加载对应的Format工厂类，将Kafka消息字节流与Flink表行数据进行相互转换。
scan.topic-partition-discovery.interval	否	Duration	5min	有效的持续时间，如30s、5min	设置动态分区发现的检查间隔。 设置后可自动发现Topic新增分区，无需重启作业。Flink运行时启动后台线程按此间隔定期查询Topic元数据，发现新分区后自动分配给空闲子任务消费。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
properties.*	否	String	-	Kafka Consumer配置	传递给Kafka Consumer的任意配置参数，前缀为properties.。 配置项名称去掉properties.前缀后直接透传给Kafka Consumer。Flink运行时在创建Kafka客户端时将这些参数原样传入，可用于调整消费行为（如fetch.min.bytes、max.poll.records等）。

Sink参数

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	kafka	标识连接器类型为Kafka连接器。 必须填写且值固定为kafka。Flink运行时根据此值加载对应的Kafka Source/Sink工厂类，将表声明为Kafka数据源或输出表。
topic	是	String	无	有效的Topic名	指定写入的Kafka主题名称。 Sink必填。Flink运行时将所有输出记录发送到此Topic，如Topic不存在需预先创建或开启Kafka自动创建Topic配置。
properties.bootstrap.servers	是	String	无	host:port列表	指定Kafka Broker地址列表，多个地址以逗号分隔。 Source和Sink均必填。Flink运行时使用此配置创建Kafka Consumer或Producer客户端连接，建议配置多个Broker地址以实现高可用。
format	是	String	无	json / csv / avro / debezium-json / canal-json / raw	指定消息的序列化格式。 Sink必填。Flink运行时根据此值将Flink表行数据序列化为对应格式的字节流写入Kafka。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
sink.partitioner	否	String	default	default / fixed / round-robin / 自定义类全限定名	指定写入Kafka的分区策略。 默认为default。default按Key哈希分区（无Key时轮询）；fixed每个Sink子任务固定写一个分区；round-robin轮询所有分区。Flink运行时根据此策略决定每条记录发送到Kafka的哪个分区，影响数据在分区间的分布均衡性。
sink.semantic	否	String	at-least-once	at-least-once / exactly-once / none	指定写入的语义保证级别。 默认为at-least-once。设置为exactly-once时需开启Flink Checkpoint机制。Flink运行时在at-least-once模式下每条记录立即提交；exactly-once模式下使用Kafka事务在Checkpoint完成时才提交数据，确保数据不重复但会增加延迟。
properties.*	否	String	-	Kafka Producer配置	传递给Kafka Producer的任意配置参数，前缀为properties.。 配置项名称去掉properties.前缀后直接透传给Kafka Producer。Flink运行时在创建Kafka Producer时将这些参数原样传入，可用于调整生产行为（如batch.size、linger.ms等）。

格式参数

以下为常用的格式相关参数，通过 format 参数指定格式后生效。

JSON格式参数

参数	是否必填	类型	默认值	取值范围	说明
json.fail-on-missing-field	否	Boolean	false	true / false	控制当JSON消息中缺少DDL定义的字段时是否报错。 默认为false。设置为true时遇到缺失字段抛出异常导致作业失败；false时将缺失字段置为null继续处理。Flink运行时在反序列化阶段检查每个字段，此参数决定了字段缺失时的容错策略。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
json.ignore-parse-errors	否	Boolean	false	true / false	控制是否忽略JSON解析错误。 默认为false。设置为true时遇到解析错误跳过该字段而非报错，可提高容错性但可能导致数据丢失。Flink运行时在反序列化遇到异常时，true模式将异常字段置为null继续处理后续字段，false模式直接抛出异常终止作业。
json.timestamp-format.standard	否	String	SQL	SQL / ISO-8601	指定时间戳类型的解析格式标准。 默认为SQL。SQL格式解析为yyyy-MM-dd HH:mm:ss.S；ISO-8601格式解析为yyyy-MM-dd'T'HH:mm:ss.S。Flink运行时根据此标准选择对应的解析器，影响JSON中时间戳字符串与Flink TIMESTAMP类型之间的转换规则。
json.map-null-key.mode	否	String	FAIL	FAIL / DROP / LITERAL	指定Map类型中null键的处理方式。 默认为FAIL。FAIL遇到null键时抛出异常；DROP丢弃null键的键值对；LITERAL将null键替换为指定字符串。Flink运行时在序列化Map数据时根据此策略处理null键，影响包含null键的Map数据的输出行为。
json.map-null-key.literal	否	String	"null"	任意字符串	指定当json.map-null-key.mode为LITERAL时null键的替换字符串。 仅在json.map-null-key.mode为LITERAL时生效。Flink运行时在序列化Map数据时，将null键替换为此参数指定的字符串值输出。

CSV格式参数

参数	是否必填	类型	默认值	取值范围	说明
csv.field-delimiter	否	String	,	单字符	指定CSV字段的分隔符。 默认为逗号（,）。仅支持单字符。Flink运行时使用此分隔符拆分或拼接CSV记录的各字段值。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
csv.disable-quote-character	否	Boolean	false	true / false	控制是否禁用引号字符。 默认为false。设置为true时引号字符将被视为普通字符处理。Flink运行时在解析CSV时，禁用引号后含分隔符的字段值不再被引号包裹保护，可能导致字段解析错误。
csv.quote-character	否	String	"	单字符	指定CSV中的引号字符，用于包裹含分隔符的字段值。 默认为双引号（"）。仅在csv.disable-quote-character为false时生效。Flink运行时在解析和生成CSV时，使用此字符识别和包裹含有分隔符或换行符的字段值。
csv.allow-comments	否	Boolean	false	true / false	控制是否允许CSV中包含注释行（以#开头）。 默认为false。设置为true时以#开头的行将被跳过。Flink运行时在解析CSV记录时，遇到#开头的行将视为注释忽略，不作为数据记录处理。
csv.ignore-parse-errors	否	Boolean	false	true / false	控制是否忽略CSV解析错误。 默认为false。设置为true时遇到解析错误的字段置为null继续处理。Flink运行时在反序列化遇到格式异常时，true模式跳过错误字段继续处理后续字段，false模式直接抛出异常终止作业。
csv.array-element-delimiter	否	String	;	任意字符串	指定数组元素的分隔符。 默认为分号（;）。Flink运行时在解析和生成CSV中的数组类型字段时，使用此分隔符拆分或拼接数组元素。

Avro格式参数

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
encoding	否	String	binary	binary / json	指定Avro的编码格式。 binary为二进制编码，json为JSON编码。Flink运行时根据此值选择binary 数据的序列化/反序列化方式，二进制编码更紧凑高效，JSON编码可读性更强。

使用示例

- **示例1：从Kafka主题消费JSON格式消息，最简配置。**

```
-- 创建Kafka源表
CREATE TABLE kafka_source (
  user_id BIGINT,
  event_type STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'kafka',
  'topic' = 'user-events',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'flink-consumer-group',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  user_id BIGINT,
  event_type STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT user_id, event_type, event_time
FROM kafka_source;
```

- **示例2：从Kafka主题的最早偏移量开始消费，适用于需要全量回放数据的场景。**

```
-- 创建Kafka源表（从最早偏移量消费）
CREATE TABLE kafka_source_earliest (
  order_id STRING,
  amount DECIMAL(10, 2),
  order_time TIMESTAMP(3)
) WITH (
  'connector' = 'kafka',
  'topic' = 'orders',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'order-consumer',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  order_id STRING,
  amount DECIMAL(10, 2),
  order_time TIMESTAMP(3)
) WITH (
```


版权所有 © 华为云计算技术有限公司

```
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT order_id, amount, order_time
FROM kafka_source_earliest;
```

- **示例3：从指定时间戳开始消费，适用于从某个时间点恢复数据处理的场景。**

```
-- 创建Kafka源表（从指定时间戳消费）
CREATE TABLE kafka_source_timestamp (
  log_id STRING,
  log_level STRING,
  message STRING,
  log_time TIMESTAMP(3)
) WITH (
  'connector' = 'kafka',
  'topic' = 'application-logs',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'log-consumer',
  'scan.startup.mode' = 'timestamp',
  'scan.startup.timestamp-millis' = '1704067200000',
  'format' = 'json'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  log_id STRING,
  log_level STRING,
  message STRING,
  log_time TIMESTAMP(3)
) WITH (
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT log_id, log_level, message, log_time
FROM kafka_source_timestamp;
```

- **示例4：消费CSV格式的Kafka消息，自定义字段分隔符。**

```
-- 创建Kafka源表（CSV格式）
CREATE TABLE kafka_csv_source (
  id INT,
  name STRING,
  score DOUBLE
) WITH (
  'connector' = 'kafka',
  'topic' = 'csv-data',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'csv-consumer',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv',
  'csv.field-delimiter' = '|'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  id INT,
  name STRING,
  score DOUBLE
) WITH (
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT id, name, score
FROM kafka_csv_source;
```

 版权所有 © 华为云计算技术有限公司

1.3 Doris 连接器

概述

Doris连接器用于读写**表格存储服务CloudTable Doris**数据库中的数据，支持通过Stream Load方式高效写入数据，并通过Doris FE的查询接口读取数据。

表 1-3 连接器详情

类别	详情
支持类型	源表、维表和结果表
运行模式	流模式和批模式
数据格式	暂不支持
API种类	SQL，Datastream
是否支持更新或删除结果表数据	是

适用场景

- **实时数据写入**：将Flink处理后的实时数据写入Doris OLAP表，支撑实时报表查询。
- **数据同步**：将Kafka等数据源的数据同步到Doris，实现实时数仓建设。
- **OLAP分析**：从Doris读取维度数据，与实时流进行关联分析。
- **ETL输出**：将Flink ETL处理结果落库到Doris，供BI工具直接查询。

约束与限制

- Exactly-Once写入需开启Flink Checkpoint，并设置唯一的sink.label-prefix。
- 写入Doris时，字段类型需与Doris表列类型兼容。
- CloudTable Doris需要为3.x及以上版本。

语法结构

Source

```
CREATE TABLE {table_name} (
  {column_name} {column_type},
  ...
) WITH (
  'connector' = 'doris',
  'fenodes' = '{fe_host:http_port}',
  'table.identifier' = '{database}.{table}',
  'username' = '{username}',
  'password' = '{password}'
);
```

Sink

版权所有 © 华为云计算技术有限公司

```
CREATE TABLE {table_name} (
  {column_name} {column_type},
  ...
) WITH (
  'connector' = 'doris',
  'fenodes' = '{fe_host:http_port}',
  'table.identifier' = '{database}.{table}',
  'username' = '{username}',
  'password' = '{password}',
  'sink.label-prefix' = '{label_prefix}',
  'sink.properties.format' = '{data_format}',
  'sink.enable-delete' = '{enable_delete}'
);
```

WITH 参数说明

通用参数

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	doris	标识连接器类型为Doris连接器。 必须填写且值固定为doris。Flink运行时根据此值加载Doris连接器工厂类，将表声明为Doris数据源或输出表。
fenodes	是	String	无	host:port列表	指定Doris FE节点HTTP地址，多个地址以逗号分隔。 Source和Sink均必填。Flink运行时使用此地址连接Doris FE获取查询计划或提交Stream Load请求，建议配置多个FE地址以实现高可用故障切换。
table.identifier	是	String	无	database.table	指定Doris表标识，格式为数据库名.表名。 Source和Sink均必填。Flink运行时根据此标识定位Doris中的目标表，影响数据的读取和写入目标。
username	是	String	无	有效的用户名	指定Doris数据库用户名。 Source和Sink均必填。Flink运行时使用此用户名进行Doris的身份认证，用户需具有对应表的读写权限。
password	是	String	无	有效的密码	指定Doris数据库密码。 Source和Sink均必填。Flink运行时使用此密码配合用户名完成Doris的身份认证。

Source参数

 版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
doris.read.field	否	String	无	列名列表	指定读取的列，多个列以逗号分隔。 未指定时读取所有列。Flink运行时将此列列表下推到Doris查询SQL的SELECT子句中，减少不必要的数据传输，提高读取效率。
doris.filter.query	否	String	无	SQL过滤条件	指定读取时的过滤条件，下推到Doris执行。 Flink运行时将此条件追加到Doris查询SQL的WHERE子句中，在Doris侧完成数据过滤，减少传输到Flink的数据量。
doris.request.tablet.size	否	Integer	2	正整数	设置每个Doris BE分区并行读取的Tablet数。 Flink运行时根据此值将Tablet分配给多个读取子任务并行执行，增大此值可提高读取并行度但增加Doris侧负载。
doris.request.connect.timeout.ms	否	Integer	30000	正整数	设置连接Doris BE的超时时间（毫秒）。 Flink运行时在建立与Doris BE的连接时，超过此时间未建立成功则抛出超时异常，网络不稳定时可适当增大。
doris.request.read.timeout.ms	否	Integer	30000	正整数	设置读取Doris BE数据的超时时间（毫秒）。 Flink运行时在从Doris BE读取数据时，超过此时间未收到响应则抛出超时异常，大查询场景可适当增大。
doris.request.query.timeout.s	否	Integer	3600	正整数	设置查询Doris的超时时间（秒）。 Flink运行时在提交查询到Doris后，超过此时间未完成则取消查询，大表全量读取场景可适当增大。
doris.request.retries	否	Integer	3	正整数	设置请求Doris的重试次数。 Flink运行时在请求失败时自动重试，重试次数耗尽后抛出异常，网络不稳定时可适当增大。
doris.deserialize.arrow.async	否	Boolean	false	true / false	控制是否异步将Arrow数据反序列化。 默认为false即同步反序列化。设置为true时Flink运行时使用独立线程池异步执行Arrow到RowData的转换，可提高读取吞吐但增加内存消耗。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
doris.deserialize.queue.size	否	Integer	64	正整数	设置异步反序列化队列大小。 仅在doris.deserialize.arrow.async为true时生效。Flink运行时在异步反序列化模式下使用此队列缓冲已反序列化的数据，队列满时反序列化线程阻塞等待。

Sink参数

参数	是否必填	类型	默认值	取值范围	说明
sink.label-prefix	否	String	无	有效的标签前缀	指定Stream Load标签前缀。 建议每个作业设置唯一前缀，用于确保Exactly-Once语义。Flink运行时使用标签前缀+唯一标识生成Stream Load的label，Doris通过label去重保证同一批数据不会重复写入。
sink.properties.format	否	String	json	json / csv	指定Stream Load数据传输格式。 默认为json。Flink运行时根据此格式将Flink行数据序列化为对应格式后发送给Doris Stream Load接口，json格式兼容性好，csv格式传输量更小。
sink.properties.max_filter_ratio	否	String	0	0~1	指定Stream Load最大过滤比率。 超过此比率的异常数据将导致整个写入批次失败。Flink运行时在Stream Load完成后检查过滤比率，超限则抛出异常触发重试，设置为0表示不允许任何过滤错误。
sink.enable-delete	否	Boolean	true	true / false	控制是否启用删除操作。 默认为false。启用后Flink的retract记录（-U、-D）将被转换为Doris的DELETE操作。Flink运行时在序列化记录时，对retract记录生成Doris的删除标记而非普通写入，仅Doris Unique Key模型表支持删除。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
sink.buffer-size	否	MemorySize	1mb	有效的内存大小，如128mb、256mb	设置Sink缓冲区大小。 Flink运行时在内存中缓冲写入数据，缓冲区满后触发Stream Load提交，增大此值可减少提交次数提高吞吐但增加内存占用和数据延迟。
sink.buffer.count	否	Integer	3	正整数	设置Sink缓冲区数量。 Flink运行时维护多个缓冲区轮流使用，当一个缓冲区正在提交时其他缓冲区继续接收数据，增大此值可提高写入并发度但增加内存占用。
sink.max-retries	否	Integer	1	正整数	设置写入失败的最大重试次数。 Flink运行时在Stream Load提交失败时自动重试，重试次数耗尽后作业失败，网络或Doris不稳定时可适当增大。
sink.check-interval	否	Duration	0s	有效的持续时间，如30s、5min	设置检查缓冲区是否需要刷新的间隔（毫秒）。 Flink运行时启动后台线程按此间隔检查缓冲区状态，满足刷新条件时触发Stream Load提交，间隔越短数据延迟越低但检查开销越大。

Lookup参数

参数	是否必填	类型	默认值	取值范围	说明
lookup.cache.max-rows	否	Long	-1	正整数	设置维表查询缓存的最大行数。 仅对Lookup JOIN场景生效。Flink运行时在内存中维护LRU缓存，缓存行数达到此上限后淘汰最久未访问的记录，增大此值可提高查询命中率但增加内存消耗。
lookup.cache.ttl	否	Duration	10s	正整数+时间单位	设置维表查询缓存的TTL（存活时间）。 仅对Lookup JOIN场景生效。Flink运行时在缓存记录超过TTL后将其失效，下次访问时重新从Doris查询加载，用于平衡数据时效性与查询性能。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
lookup.max-retries	否	Integer	1	正整数	设置维表查询的最大重试次数。 Flink运行时在查询Doris失败时自动重试，重试次数耗尽后抛出异常，Doris负载较高时可适当增大。
jdbc-url	是	String	无	无	连接Doris的MySQL协议地址。

使用示例

- **示例1：从Doris表中批量读取数据。**

```
-- 创建Doris源表
CREATE TABLE doris_source (
  user_id BIGINT,
  user_name STRING,
  age INT,
  city STRING
) WITH (
  'connector' = 'doris',
  'fenodes' = 'doris-fe:8030',
  'table.identifier' = 'user_db.user_info',
  'username' = 'root',
  'password' = 'xxxx'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  user_id BIGINT,
  user_name STRING,
  age INT,
  city STRING
) WITH (
  'connector' = 'print',
  'print-identifier' = 'doris-data'
);

-- 提交作业
INSERT INTO print_sink
SELECT user_id, user_name, age, city
FROM doris_source;
```

- **示例2：将数据以JSON格式写入Doris表。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  user_id BIGINT,
  event_type STRING,
  event_time TIMESTAMP(3),
  event_count BIGINT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.user_id.kind' = 'sequence',
  'fields.user_id.start' = '1',
  'fields.user_id.end' = '100000',
  'fields.event_count.kind' = 'sequence',
  'fields.event_count.start' = '1',
  'fields.event_count.end' = '100'
);
```

 版权所有 © 华为云计算技术有限公司

```
-- 创建Doris结果表
CREATE TABLE doris_sink (
  user_id BIGINT,
  event_type STRING,
  event_time TIMESTAMP(3),
  event_count BIGINT
) WITH (
  'connector' = 'doris',
  'fenodes' = 'doris-fe:8030',
  'table.identifier' = 'analytics_db.user_events',
  'username' = 'root',
  'password' = 'xxxx',
  'sink.label-prefix' = 'flink-user-events',
  'sink.properties.format' = 'json',
  'sink.max-retries' = '3'
);

-- 提交作业
INSERT INTO doris_sink
SELECT user_id, event_type, event_time, event_count
FROM datagen_source;
```

- **示例3：从Kafka消费实时事件数据，聚合计算后写入Doris，支撑实时OLAP查询。**

```
-- Kafka源表
CREATE TABLE kafka_event_source (
  user_id BIGINT,
  event_type STRING,
  page_url STRING,
  event_time TIMESTAMP(3),
  WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
) WITH (
  'connector' = 'kafka',
  'topic' = 'user-events',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'doris-writer',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json',
  'json.ignore-parse-errors' = 'true'
);

-- Doris结果表
CREATE TABLE doris_event_sink (
  window_start TIMESTAMP(3),
  window_end TIMESTAMP(3),
  event_type STRING,
  event_count BIGINT,
  unique_users BIGINT
) WITH (
  'connector' = 'doris',
  'fenodes' = 'doris-fe:8030',
  'table.identifier' = 'analytics_db.event_stats',
  'username' = 'root',
  'password' = 'xxxx',
  'sink.label-prefix' = 'flink-event-stats',
  'sink.properties.format' = 'json',
  'sink.enable-delete' = 'false'
);

-- 5分钟窗口聚合写入
INSERT INTO doris_event_sink
SELECT
  window_start AS window_start,
  window_end AS window_end,
  event_type,
  COUNT(*) AS event_count,
  COUNT(DISTINCT user_id) AS unique_users
FROM TABLE(
  HOP(TABLE kafka_event_source, DESCRIPTOR(event_time), INTERVAL '1' MINUTE, INTERVAL '5'
```

 版权所有 © 华为云计算技术有限公司

```
MINUTE)
)
GROUP BY window_start, window_end, event_type;
```

- **示例4：将Doris表作为维表，通过Lookup JOIN关联Kafka实时流。**

```
-- Doris维表
CREATE TABLE doris_dim (
  user_id BIGINT,
  user_name STRING,
  city STRING,
  PRIMARY KEY (user_id) NOT ENFORCED
) WITH (
  'connector' = 'doris',
  'fenodes' = 'doris-fe:8030',
  'table.identifier' = 'user_db.user_info',
  'jdbc-url' = 'jdbc:mysql://FE实例IP地址:FE查询连接端口',
  'username' = 'root',
  'password' = 'xxxx',
  'lookup.cache.max-rows' = '10000',
  'lookup.cache.ttl' = '10min'
);

-- Kafka实时流
CREATE TABLE kafka_realtime_source (
  user_id BIGINT,
  event_type STRING,
  event_time TIMESTAMP(3),
  proctime AS PROCTIME()
) WITH (
  'connector' = 'kafka',
  'topic' = 'events',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'doris-lookup',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  user_id BIGINT,
  user_name STRING,
  city STRING,
  event_type STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'print',
  'print-identifier' = 'doris-lookup-result'
);

-- 提交作业：Lookup JOIN后输出
INSERT INTO print_sink
SELECT e.user_id, d.user_name, d.city, e.event_type, e.event_time
FROM kafka_realtime_source e
LEFT JOIN doris_dim FOR SYSTEM_TIME AS OF e.proctime AS d
ON e.user_id = d.user_id;
```

- **示例5：启用Doris Sink的删除能力，将Flink的回撤记录转换为Doris删除操作。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  user_id BIGINT,
  user_name STRING,
  city STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.user_id.kind' = 'sequence',
  'fields.user_id.start' = '1',
  'fields.user_id.end' = '10000',
  'fields.user_name.length' = '10'
);
```

 版权所有 © 华为云计算技术有限公司

```
-- 创建Doris Upsert结果表
CREATE TABLE doris_upsert_sink (
  user_id BIGINT,
  user_name STRING,
  city STRING,
  PRIMARY KEY (user_id) NOT ENFORCED
) WITH (
  'connector' = 'doris',
  'fenodes' = 'doris-fe:8030',
  'table.identifier' = 'user_db.user_info',
  'username' = 'root',
  'password' = 'xxxx',
  'sink.label-prefix' = 'flink-user-upsert',
  'sink.properties.format' = 'json',
  'sink.enable-delete' = 'true'
);

-- 提交作业
INSERT INTO doris_upsert_sink
SELECT user_id, user_name, city
FROM datagen_source;
```

1.4 Upsert-Kafka 连接器

概述

Upsert-Kafka连接器是Kafka连接器的变更日志（Changelog）变体，用于以Upsert方式读写**DMS Kafka主题（Topic）**。与普通Kafka连接器仅处理追加（Append）数据不同，Upsert-Kafka连接器将Kafka消息解释为变更日志记录，支持按主键的插入、更新和删除操作，适用于需要端到端Exactly-Once语义的CDC（Change Data Capture）场景。

类别	详情
支持类型	源表和结果表
运行模式	流模式
数据格式	详情请参见**Formats**。
API种类	SQL，Datastream
是否支持更新或删除结果表数据	是

适用场景

- **CDC数据管道**：将Debezium等CDC工具捕获的变更数据以Upsert方式写入Kafka，下游消费时自动按主键合并为最新状态。
- **实时数仓同步**：将Flink聚合计算产生的回撤记录（-U/+U）写入Kafka，下游消费者可获取完整的变更语义。
- **跨系统数据同步**：作为不同系统间的变更数据传输通道，保证数据一致性。
- **事件溯源**：将领域事件以变更日志形式持久化到Kafka，支持状态重建和时间旅行查询。

版权所有 © 华为云计算技术有限公司

- **维度表广播**：将维度表的变更数据广播到Kafka，下游作业按主键更新本地维度缓存。

约束与限制

- 与普通Kafka连接器（connector = 'kafka'）互不兼容，同一Topic不建议混用两种连接器读写。
- 默认消费数据时，topic须提前创建。
- DMS Kafka需要为3.x及以上版本。

与普通 Kafka 连接器的区别

对比项	Kafka连接器	Upsert-Kafka连接器
connector值	kafka	upsert-kafka
主键要求	不需要	必须定义PRIMARY KEY
数据语义	追加流（Append-only）	变更日志流（Changelog）
Source输出	原始消息	按主键去重后的最新状态
Sink写入	仅追加写入	按主键Upsert写入（更新或插入）
删除支持	不支持	Sink支持删除（-D记录写为NULL值）
适用场景	事件流、日志采集	CDC同步、状态更新、维度广播
消息格式	普通键值对	Key为主键字段值，Value为完整记录或NULL（删除）

语法结构

```
CREATE TABLE {table_name} (
  {column_name} {column_type},
  ...
  PRIMARY KEY ({pk_column}) NOT ENFORCED,
  WATERMARK FOR {time_column} AS {watermark_expression}
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = '{kafka_topic}',
  'properties.bootstrap.servers' = '{broker_list}',
  'properties.group.id' = '{consumer_group}',
  'key.format' = '{key_format}',
  'value.format' = '{value_format}'
);
```

说明

Upsert-Kafka连接器必须分别指定Key和Value的序列化格式。Key对应主键字段，Value对应非主键字段。删除记录时Value为NULL。

版权所有 © 华为云计算技术有限公司

WITH 参数说明

Source参数

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	upsert-kafka	标识连接器类型为Upsert-Kafka连接器。 必须填写且值固定为upsert-kafka。Flink运行时根据此值加载Upsert-Kafka Source工厂类，将表声明为变更日志数据源，按主键对消息进行去重合并。
topic	是	String	无	有效的Topic名	指定消费的Kafka主题名称。 Source必填。Flink运行时为每个Topic分区分配一个子任务并行消费，主键相同的记录仅保留最新值。
properties.bootstrap.servers	是	String	无	host:port列表	指定Kafka Broker地址列表，多个地址以逗号分隔。 Source和Sink均必填。Flink运行时使用此配置创建Kafka Consumer或Producer客户端连接，建议配置多个Broker地址以实现高可用。
properties.group.id	否	String	无	有效的消费者组名	指定Kafka消费者组ID。 Source必填。Flink运行时使用此组ID标识消费者身份，相同组ID的消费者共享Topic分区的消费进度。
key.format	是	String	无	json / csv / avro	指定主键字段的反序列化格式。 Source和Sink均必填。Flink运行时根据此格式将Kafka消息的Key部分（主键字段）进行反序列化。主键字段的序列化/反序列化必须使用与写入时相同的格式。
value.format	是	String	无	json / csv / avro	指定非主键字段的反序列化格式。 Source和Sink均必填。Flink运行时根据此格式将Kafka消息的Value部分（非主键字段）进行反序列化。删除记录的Value为NULL。
scan.topic-partition-discovery.interval	否	Duration	无	正整数（毫秒）	设置动态分区发现的检查间隔。 设置后可自动发现Topic新增分区。
properties.*	否	String	-	Kafka Consumer配置	传递给Kafka Consumer的任意配置参数，前缀为properties.。 配置项名称去掉properties.前缀后直接透传给Kafka Consumer。

版权所有 © 华为云计算技术有限公司

Sink参数

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	upsert-kafka	标识连接器类型为Upsert-Kafka连接器。 必须填写且值固定为upsert-kafka。Flink运行时根据此值加载Upsert-Kafka Sink工厂类，将表声明为变更日志输出表，按主键进行Upsert写入。
topic	是	String	无	有效的Topic名	指定写入的Kafka主题名称。 Sink必填。Flink运行时将所有输出记录发送到此Topic，Key为主键字段值，Value为完整记录或NULL（删除记录）。
properties.bootstrap.servers	是	String	无	host:port列表	指定Kafka Broker地址列表，多个地址以逗号分隔。Source和Sink均必填。
key.format	是	String	无	json / csv / avro	指定主键字段的序列化格式。 Sink必填。Flink运行时根据此格式将主键字段序列化为Kafka消息的Key部分，用于分区路由和主键标识。
value.format	是	String	无	json / csv / avro	指定非主键字段的序列化格式。 Sink必填。Flink运行时根据此格式将非主键字段序列化为Kafka消息的Value部分。删除记录时Value序列化为NULL。
sink.buffer-flush.max-rows	否	Integer	无	正整数	设置Sink缓冲区最大行数。 缓冲区满后触发批量写入。增大此值可减少写入次数提高吞吐但增加延迟。仅对at-least-once和none语义生效。
sink.buffer-flush.interval	否	Duration	无	有效的持续时间，如10s,1min	设置Sink缓冲区刷新间隔。支持 每隔此时间触发一次批量写入。与sink.buffer-flush.max-rows共同控制刷新策略，任一条件满足即触发。仅对at-least-once和none语义生效。
properties.*	否	String	-	Kafka Producer配置	传递给Kafka Producer的任意配置参数，前缀为properties.。 配置项名称去掉properties.前缀后直接透传给Kafka Producer。

Key和Value格式参数

版权所有 © 华为云计算技术有限公司

Upsert-Kafka连接器需要分别指定Key和Value的格式，各自有独立的格式参数，前缀分别为 key. 和 value.。

JSON格式参数（以value为例，key同理替换前缀）

参数	是否必填	类型	默认值	取值范围	说明
value.json.fail-on-missing-field	否	Boolean	false	true / false	控制当JSON消息中缺少DDL定义的字段时是否报错。 默认为false。设置为true时遇到缺失字段抛出异常；false时将缺失字段置为null继续处理。
value.json.ignore-parse-errors	否	Boolean	false	true / false	控制是否忽略JSON解析错误。 默认为false。设置为true时遇到解析错误跳过该字段而非报错，可提高容错性但可能导致数据丢失。
value.json.timestamp-format.standard	否	String	SQL	SQL / ISO-8601	指定时间戳类型的解析格式标准。 默认为SQL。SQL格式解析为yyyy-MM-dd HH:mm:ss.S；ISO-8601格式解析为yyyy-MM-dd'T'HH:mm:ss.S。

CSV格式参数（以value为例）

参数	是否必填	类型	默认值	取值范围	说明
value.csv.field-delimiter	否	String	,	单字符	指定CSV字段的分隔符。默认为逗号（,）。
value.csv.ignore-parse-errors	否	Boolean	false	true / false	控制是否忽略CSV解析错误。默认为false。

使用示例

- **示例1：从Upsert-Kafka主题消费变更日志数据，按主键自动去重合并，输出到Print结果表。**

```
-- 创建Upsert-Kafka源表（JSON格式）
CREATE TABLE upsert_kafka_order_source (
  order_id STRING,
  user_id BIGINT,
  amount DECIMAL(10, 2),
  status STRING,
```


版权所有 © 华为云计算技术有限公司

```
  update_time TIMESTAMP(3),
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'order-changes',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'order-consumer',
  'key.format' = 'json',
  'value.format' = 'json'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  order_id STRING,
  user_id BIGINT,
  amount DECIMAL(10, 2),
  status STRING,
  update_time TIMESTAMP(3)
) WITH (
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT order_id, user_id, amount, status, update_time
FROM upsert_kafka_order_source;
```

- **示例2：Key使用CSV格式，Value使用JSON格式，适用于主键字段较少的场景。**

```
-- 创建Upsert-Kafka源表（Key=CSV，Value=JSON）
CREATE TABLE upsert_kafka_mixed_source (
  user_id BIGINT,
  user_name STRING,
  city STRING,
  member_level STRING,
  update_time TIMESTAMP(3),
  PRIMARY KEY (user_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'user-profile-changes',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'profile-consumer',
  'key.format' = 'csv',
  'value.format' = 'json'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  user_id BIGINT,
  user_name STRING,
  city STRING,
  member_level STRING,
  update_time TIMESTAMP(3)
) WITH (
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT user_id, user_name, city, member_level, update_time
FROM upsert_kafka_mixed_source;
```

- **示例3：Upsert-Kafka端到端CDC管道，将变更数据从一个主题转发到另一个主题。**

```
-- 上游CDC源表
CREATE TABLE upsert_kafka_cdc_source (
  order_id STRING,
  user_id BIGINT,
  amount DECIMAL(10, 2),
  status STRING,
  PRIMARY KEY (order_id) NOT ENFORCED
```

 版权所有 © 华为云计算技术有限公司

```
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'cdc-order-changes',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'cdc-relay-consumer',
  'key.format' = 'json',
  'value.format' = 'json'
);

-- 下游转发目标表
CREATE TABLE upsert_kafka_cdc_target (
  order_id STRING,
  user_id BIGINT,
  amount DECIMAL(10, 2),
  status STRING,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'cdc-order-relay',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'key.format' = 'json',
  'value.format' = 'json'
);

-- 变更日志转发
INSERT INTO upsert_kafka_cdc_target
SELECT order_id, user_id, amount, status
FROM upsert_kafka_cdc_source;
```

1.5 FileSystem 连接器

概述

FileSystem连接器用于在AI DataLake的Flink作业中实现与文件系统的数据交互。该连接器支持对**本地文件系统**、**对象存储服务OBS**等存储后端的批流统一读写，提供多文件格式和分区写入能力，适用于数据归档、文件导出和批处理ETL等场景。

表 1-4 连接器详情

类别	详情
支持类型	源表和结果表
数据格式	CSV，JSON，Avro，Parquet，Orc，Debezium-JSON，Canal-JSON，Raw
API种类	SQL，Datastream
是否支持更新或删除结果表数据	否

适用场景

- **数据归档**：将流式数据按时间分区滚动写入HDFS或OBS，实现长期归档存储。
- **文件导出**：将处理结果以CSV或JSON格式导出到文件系统，供下游系统消费。
- **批处理ETL**：从文件系统批量读取数据，清洗转换后写入目标存储。

版权所有 © 华为云计算技术有限公司

- **列式存储优化**：以Parquet或ORC格式写入文件，提升下游查询性能。
- **多源数据汇聚**：将多个数据源的数据统一写入文件系统，构建数据湖。

约束与限制

- 文件格式需与format参数配置一致，否则读取时会报错。
- Exactly-Once语义依赖Checkpoint机制，需确保Checkpoint正常触发且间隔合理。

语法结构

Source

```
CREATE TABLE fs_source (
  {column_name} {column_type},
  ...
) WITH (
  'connector' = 'filesystem',
  'path' = '{file_system_path}',
  'format' = '{file_format}'
);
```

Sink

```
CREATE TABLE fs_sink (
  {column_name} {column_type},
  ...
) PARTITIONED BY ({partition_columns}) WITH (
  'connector' = 'filesystem',
  'path' = '{file_system_path}',
  'format' = '{file_format}',
  'sink.rolling-policy.file-size' = '{size}',
  'sink.rolling-policy.rollover-interval' = '{interval}'
);
```

WITH 参数说明

公共参数

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	filesystem	指定连接器类型为FileSystem连接器。 该参数为固定值，不可修改。配置时必须设置为filesystem，系统根据此值加载对应的文件系统连接器实现。
path	是	String	无	有效的文件系统路径	指定文件系统的读写路径。 支持本地路径（file:///）、OBS路径（obs://）等。配置时需确保Flink作业对该路径有读写权限。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
format	是	String	无	csv、json、avro、parquet、orc	指定文件的序列化格式。 ● csv为逗号分隔的文本格式 ● json为JSON行格式 ● avro为Apache Avro行式格式 ● parquet为列式存储格式 ● orc为Optimized Row Columnar列式格式 配置时需根据数据特性和查询需求选择。

Source专有参数

参数	是否必填	类型	默认值	取值范围	说明
source.monitor-interval	否	Duration	无	有效的持续时间，如10s、1min	指定流式模式下监控目录新文件的时间间隔。 配置后Source将周期性扫描目录，发现新文件时自动读取。未配置时为批式读取模式，仅读取一次目录下所有文件。配置时需根据文件生成频率调整扫描间隔。

Sink专有参数

参数	是否必填	类型	默认值	取值范围	说明
sink.rolling-policy.file-size	否	MemorySize	128 MB	有效的内存大小，如64mb、256mb	指定滚动文件的目标大小。 当单个文件达到该大小时滚动生成新文件。配置时需在文件数量和单文件大小之间权衡，文件过小产生过多小文件影响查询性能，文件过大影响写入延迟。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
sink.rolling-policy.rollover-interval	否	Duration	30min	有效的持续时间，如30s、5min	指定滚动文件的时间间隔。 当文件打开时间超过该间隔时滚动生成新文件。配置时需根据数据可见性需求调整，间隔过短产生过多小文件，间隔过长数据延迟高。
sink.rolling-policy.check-interval	否	Duration	60s	有效的持续时间，如30s、5min	基于时间的滚动策略检查间隔。 该参数用于控制检查部分文件（part file）是否应根据 sink.rolling-policy.rollover-interval 参数进行滚动的频率。
sink.partition-commit.policy.kind	否	String	无	success-file、metadata、success-file+metadata	指定分区提交策略。 ● success-file表示提交时创建_SUCCESS标记文件。 ● metadata表示提交时写入分区元数据。 配置后系统在分区数据写入完成后触发提交动作，通知下游分区已就绪。
sink.partition-commit.delay	否	Duration	0s	有效的持续时间，如1min、10min	指定分区提交的延迟时间。 分区数据写入完成后等待该时间再提交，用于确保延迟到达的数据写入正确分区。配置时需根据数据延迟情况设定，延迟过短可能导致数据写入错误分区。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
sink.partition-commit.trigger	否	String	process-time	process-time、partition-time	指定分区提交的触发器类型。 ● process-time表示基于系统处理时间触发提交。 ● partition-time表示基于分区时间（Watermark）触发提交。 配置时建议使用partition-time以保证数据一致性。
auto-compaction	否	Boolean	false	true、false	指定是否启用自动文件合并。 启用后系统在Checkpoint提交时自动合并小文件为目标大小的文件。配置时建议在滚动间隔较短产生大量小文件的场景下开启。
compaction.file-size	否	MemorySize	无	有效的内存大小，如128 mb、256 mb	指定文件合并后的目标文件大小。 仅在auto-compaction为true时生效。配置时需与sink.rolling-policy.file-size协调，建议设置为file-size的1至2倍。

使用示例

- **示例1：批式读取HDFS上的CSV文件，输出到Print结果表。**

```
-- 创建FileSystem源表（CSV格式，批式读取）
CREATE TABLE fs_csv_source (
  order_id BIGINT,
  user_id BIGINT,
  amount DOUBLE,
  order_time TIMESTAMP(3)
) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://xxx/data/orders/',
  'format' = 'csv'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  order_id BIGINT,
  user_id BIGINT,
  amount DOUBLE,
  order_time TIMESTAMP(3)
) WITH (
  'connector' = 'print',
  'print-identifier' = 'fs_csv_orders'
);

-- 提交作业
INSERT INTO print_sink
```

 版权所有 © 华为云计算技术有限公司

```
SELECT order_id, user_id, amount, order_time
FROM fs_csv_source;
```

- **示例2：使用DataGen生成模拟数据，以Parquet列式格式按天分区写入HDFS。**

```
-- 创建DataGen源表
CREATE TABLE event_source (
  event_id BIGINT,
  event_type STRING,
  event_time TIMESTAMP(3),
  dt STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '200',
  'fields.event_id.kind' = 'sequence',
  'fields.event_id.start' = '1',
  'fields.event_id.end' = '1000000',
  'fields.event_type.length' = '10',
  'fields.dt.kind' = 'sequence',
  'fields.dt.start' = '20260101',
  'fields.dt.end' = '20260131'
);

-- 创建FileSystem Sink结果表（Parquet格式，按天分区）
CREATE TABLE fs_parquet_sink (
  event_id BIGINT,
  event_type STRING,
  event_time TIMESTAMP(3),
  dt STRING
) PARTITIONED BY (dt) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://xxx/data/events/',
  'format' = 'parquet',
  'sink.rolling-policy.file-size' = '256mb',
  'sink.rolling-policy.rollover-interval' = '5min',
  'sink.partition-commit.policy.kind' = 'success-file',
  'sink.partition-commit.trigger' = 'partition-time'
);

-- 提交作业
INSERT INTO fs_parquet_sink
SELECT event_id, event_type, event_time, dt
FROM event_source;
```

- **示例3：流式监控HDFS目录下新增的JSON文件，实时输出到Print结果表。**

```
-- 创建FileSystem源表（JSON格式，流式监控）
CREATE TABLE fs_json_source (
  log_id BIGINT,
  log_level STRING,
  log_message STRING,
  log_time TIMESTAMP(3)
) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://xxx/data/logs/',
  'format' = 'json',
  'source.monitor-interval' = '10s'
);

-- 创建Print结果表
CREATE TABLE log_print_sink (
  log_id BIGINT,
  log_level STRING,
  log_message STRING,
  log_time TIMESTAMP(3)
) WITH (
  'connector' = 'print',
  'print-identifier' = 'fs_json_logs'
);

-- 提交作业
INSERT INTO log_print_sink
```

 版权所有 © 华为云计算技术有限公司

```
SELECT log_id, log_level, log_message, log_time
FROM fs_json_source;
```

- **示例4：从Kafka消费订单数据，以ORC列式格式按小时分区写入HDFS，启用自动文件合并。**

```
-- 创建Kafka源表
CREATE TABLE kafka_order_source (
  order_id STRING,
  user_id BIGINT,
  amount DECIMAL(10, 2),
  status STRING,
  event_time TIMESTAMP(3),
  WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND,
  dt STRING,
  hh STRING
) WITH (
  'connector' = 'kafka',
  'topic' = 'orders',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'properties.group.id' = 'fs_orc_group',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json',
  'json.ignore-parse-errors' = 'true'
);

-- 创建FileSystem Sink结果表（ORC格式，按天+小时分区）
CREATE TABLE fs_orc_sink (
  order_id STRING,
  user_id BIGINT,
  amount DECIMAL(10, 2),
  status STRING,
  event_time TIMESTAMP(3),
  dt STRING,
  hh STRING
) PARTITIONED BY (dt, hh) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://xxx/data/orders_orc/',
  'format' = 'orc',
  'sink.rolling-policy.file-size' = '128mb',
  'sink.rolling-policy.rollover-interval' = '1min',
  'sink.partition-commit.policy.kind' = 'success-file',
  'sink.partition-commit.trigger' = 'partition-time',
  'auto-compaction' = 'true',
  'compaction.file-size' = '256mb'
);

-- 提交ETL作业
INSERT INTO fs_orc_sink
SELECT order_id, user_id, amount, status, event_time,
       DATE_FORMAT(event_time, 'yyyy-MM-dd') AS dt,
       DATE_FORMAT(event_time, 'HH') AS hh
FROM kafka_order_source;
```

- **示例5：批式读取HDFS上的Avro文件，过滤后以JSON格式写入另一个HDFS目录。**

```
-- 创建FileSystem源表（Avro格式）
CREATE TABLE fs_avro_source (
  user_id BIGINT,
  user_name STRING,
  age INT,
  city STRING,
  register_time TIMESTAMP(3)
) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://xxx/data/users_avro/',
  'format' = 'avro'
);

-- 创建FileSystem Sink结果表（JSON格式）
CREATE TABLE fs_json_sink (
```

 版权所有 © 华为云计算技术有限公司

```
  user_id BIGINT,
  user_name STRING,
  age INT,
  city STRING,
  register_time TIMESTAMP(3)
) WITH (
  'connector' = 'filesystem',
  'path' = 'obs://xxx/data/users_json/',
  'format' = 'json',
  'sink.rolling-policy.file-size' = '128mb',
  'sink.rolling-policy.rollover-interval' = '5min'
);

-- 提交作业：过滤成年用户
INSERT INTO fs_json_sink
SELECT user_id, user_name, age, city, register_time
FROM fs_avro_source
WHERE age >= 18;
```

- **示例6：使用DataGen生成模拟数据，以CSV格式写入本地文件系统，适用于本地测试场景。**

```
-- 创建DataGen源表
CREATE TABLE metric_source (
  metric_id BIGINT,
  metric_name STRING,
  metric_value DOUBLE,
  metric_time TIMESTAMP(3)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.metric_id.kind' = 'sequence',
  'fields.metric_id.start' = '1',
  'fields.metric_id.end' = '100000',
  'fields.metric_name.length' = '15',
  'fields.metric_value.min' = '0.0',
  'fields.metric_value.max' = '100.0'
);

-- 创建FileSystem Sink结果表（CSV格式，本地文件系统）
CREATE TABLE fs_local_csv_sink (
  metric_id BIGINT,
  metric_name STRING,
  metric_value DOUBLE,
  metric_time TIMESTAMP(3)
) WITH (
  'connector' = 'filesystem',
  'path' = 'file:///tmp/flink/metrics/',
  'format' = 'csv',
  'sink.rolling-policy.file-size' = '64mb',
  'sink.rolling-policy.rollover-interval' = '30s'
);

-- 提交作业
INSERT INTO fs_local_csv_sink
SELECT metric_id, metric_name, metric_value, metric_time
FROM metric_source;
```

1.6 DataGen 连接器

概述

DataGen连接器用于按规则生成随机或序列测试数据，无需依赖任何外部数据源，是Flink内置的Source连接器。

 版权所有 © 华为云计算技术有限公司

表 1-5 连接器详情

类别	详情
支持类型	源表
运行模式	批模式和流模式
数据格式	暂不适用
API种类	SQL
是否支持更新或删除结果表数据	不涉及

适用场景

- **功能验证**：在无外部数据源时，快速验证Flink SQL作业逻辑。
- **性能压测**：通过控制数据生成速率和字段大小，模拟高吞吐量场景。
- **开发调试**：搭配Print/BlackHole连接器，构建零依赖的端到端调试链路。

约束与限制

- 当同时设置number-of-rows和序列生成器的start/end时，任一条件先满足即停止生成。

语法结构

```
CREATE TABLE {table_name} (
  {column_name} {column_type},
  ...
  WATERMARK FOR {time_column} AS {watermark_expression}
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '{生成速率}',
  'number-of-rows' = '{总行数}',
  'fields.{field_name}.kind' = 'random / sequence',
  'fields.{field_name}.min' = '{最小值}',
  'fields.{field_name}.max' = '{最大值}',
  'fields.{field_name}.length' = '{字符串/集合长度}',
  'fields.{field_name}.start' = '{起始值}',
  'fields.{field_name}.end' = '{结束值}'
  ...
);
```

说明

参数中的 {field_name} 需替换为DDL中定义的实际字段名。例如字段名为user_id，则参数写为fields.user_id.kind。

WITH 参数说明

通用参数

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	datagen	指定连接器类型为DataGen，必须固定填写datagen。 这是Flink SQL识别连接器实现的唯一标识，填错会导致表注册失败。
rows-per-second	否	Long	10000	正整数	控制每秒生成数据的行数。 仅对Source生效。Flink运行时每个并行子任务独立按此速率向下游发射数据，实际总吞吐量 = 此值 × 并行度。
number-of-rows	否	Long	无（无界流）	正整数	控制生成数据的总行数，设置后Source在达到指定行数时自动停止，变为有界流。 与序列生成器的 end 值独立生效，任一条件先满足即停止。未设置时Source持续生成数据，不会自动停止。

生成策略参数

参数	是否必填	类型	默认值	取值范围	说明
fields.{field_name}.kind	否	String	random	random / sequence	指定字段的数据生成策略。 ● random使用随机生成器，每次产生随机值。 ● sequence使用序列生成器，从start递增到end。 未显式设置时所有字段默认使用随机生成器。同一个表中不同字段可以分别指定不同的生成策略。

随机生成器参数（kind = random）

参数	是否必填	类型	默认值	取值范围	说明
fields.{field_name}.min	否	与字段类型一致	见类型默认值表	数值	指定随机生成的最小值。 仅对数值类型（TINYINT/SMALLINT/INT/BIGINT/FLOAT/DOUBLE/DECIMAL）有效，对STRING、BOOLEAN等类型无效。Flink在生成随机值时以此值作为下界。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
fields.{field_name}.max	否	与字段类型一致	见类型默认值表	数值	指定随机生成的最大值。 仅对数值类型。min必须小于max，否则生成行为未定义。与min配合确定随机值的取值区间。
fields.{field_name}.length	否	Integer	100	正整数	控制随机生成的字符串长度（字符数）或集合元素个数。 适用于STRING/VARCHAR/CHAR/BYTES/BINARY/VARBINARY类型控制字符串/字节长度，适用于ARRAY/MAP/MULTISET类型控制集合元素个数。值越大单条记录占用内存越多，过大会影响吞吐量和GC性能。
fields.{field_name}.max-past	否	与字段类型一致	0	有效的持续时间，如30s、5min	为TIMESTAMP类型生成"距当前时间不超过N的随机过去时间"。 只支持TIMESTAMP类型。
fields.{field_name}.null-rate	否	Float	无	大于等于0且小于等于1	控制该字段生成null值的概率。
fields.{field_name}.var-len	否	BOOLEAN	false	true/false	是控制VARCHAR/STRING/VARBINARY/BYTES生成长度是否可变的开关。

随机生成器各类型默认范围：

数据类型	默认最小值	默认最大值	可配置参数
BOOLEAN	-	-	-
TINYINT	-128	127	min, max

版权所有 © 华为云计算技术有限公司

数据类型	默认最小值	默认最大值	可配置参数
SMALLINT	-32768	32767	min, max
INT	-2147483648	2147483647	min, max
BIGINT	-9223372036854775808	9223372036854775807	min, max
FLOAT	最小正值	最大正值	min, max
DOUBLE	最小正值	最大正值	min, max
DECIMAL(p, s)	由精度决定	由精度决定	min, max
STRING / VARCHAR / CHAR	-	-	length
BYTES / BINARY / VARBINARY	-	-	length
DATE	1970-01-01	当前日期	min, max
TIME	00:00:00	23:59:59	min, max
TIMESTAMP	1970-01-01 00:00:00.000	当前时间戳	min, max
TIMESTAMP_LTZ	1970-01-01 00:00:00.000	当前时间戳	min, max
ARRAY<T>	-	-	length（元素个数）
MAP<K, V>	-	-	length（键值对个数）
MULTISET<T>	-	-	length（元素个数）
ROW<...>	-	-	各子字段独立配置

序列生成器参数（kind = sequence）

参数	是否必填	类型	默认值	取值范围	说明
fields.{field_name}.start	是	与字段类型一致	无	数值类型范围	指定序列的起始值。 仅在使用序列生成器时需要设置。 Flink从此值开始按步长1递增生成数据。

版权所有 © 华为云计算技术有限公司

参数	是否必填	类型	默认值	取值范围	说明
fields.{field_name}.end	是	与字段类型一致	无	数值类型范围	指定序列的结束值。 仅在使用序列生成器时需要设置。当序列达到此值后Source停止发送数据，变为有界流。在多并行度下，序列范围会在各子任务间均匀分配。

说明

- **序列生成器支持的数据类型：** TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL、CHAR、VARCHAR、STRING。
- CHAR、VARCHAR、STRING是将整数转为STRING类型的数字字符串。
- BOOLEAN、DATE、TIMESTAMP等类型不支持序列生成器。
- 序列生成器总是在达到end值后变为有界流。

使用示例

- **示例1：快速创建一个测试表，所有字段使用默认随机生成器，无需任何外部依赖。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  id BIGINT,
  data STRING,
  ts TIMESTAMP(3),
  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND
) WITH (
  'connector' = 'datagen'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  id BIGINT,
  data STRING,
  ts TIMESTAMP(3)
) WITH (
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT id, data, ts
FROM datagen_source;
```

- **示例2：模拟用户行为数据，ID字段使用序列递增，其他字段控制生成范围。**

```
-- 创建DataGen源表
CREATE TABLE user_behavior_source (
  user_id INT,
  user_name STRING,
  score DOUBLE,
  age INT,
  event_time TIMESTAMP(3),
  WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
) WITH (
  'connector' = 'datagen',
```

 版权所有 © 华为云计算技术有限公司

```
  'rows-per-second' = '500',
  'fields.user_id.kind' = 'sequence',
  'fields.user_id.start' = '1',
  'fields.user_id.end' = '1000',
  'fields.user_name.length' = '10',
  'fields.score.min' = '0.0',
  'fields.score.max' = '100.0',
  'fields.age.min' = '18',
  'fields.age.max' = '65'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  user_id INT,
  user_name STRING,
  score DOUBLE,
  age INT,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'print',
  'print-identifier' = 'user-behavior'
);

-- 提交作业
INSERT INTO print_sink
SELECT user_id, user_name, score, age, event_time
FROM user_behavior_source;
```

- **示例3：生成固定行数的测试数据，数据生成完毕后Source自动停止，适用于批处理测试。**

```
-- 创建DataGen源表（有界流）
CREATE TABLE datagen_bounded_source (
  id INT,
  data STRING
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'number-of-rows' = '500',
  'fields.id.kind' = 'sequence',
  'fields.id.start' = '1',
  'fields.id.end' = '500'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  id INT,
  data STRING
) WITH (
  'connector' = 'print',
  'print-identifier' = 'bounded-output'
);

-- 提交作业
INSERT INTO print_sink
SELECT id, data
FROM datagen_bounded_source;
```

- **示例4：生成包含ARRAY、MAP、ROW等复合类型字段的测试数据。**

```
-- 创建DataGen源表（含复合类型字段）
CREATE TABLE datagen_complex_source (
  id INT,
  tags ARRAY<STRING>,
  properties MAP<STRING, STRING>,
  address ROW<city STRING, street STRING>,
  payload BYTES
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.id.kind' = 'sequence',
  'fields.id.start' = '1',
```

 版权所有 © 华为云计算技术有限公司

```
  'fields.id.end' = '10000',
  'fields.tags.length' = '3',
  'fields.properties.length' = '5',
  'fields.payload.length' = '50'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  id INT,
  tags ARRAY<STRING>,
  properties MAP<STRING, STRING>,
  address ROW<city STRING, street STRING>,
  payload BYTES
) WITH (
  'connector' = 'print',
  'print-identifier' = 'complex-data'
);

-- 提交作业
INSERT INTO print_sink
SELECT id, tags, properties, address, payload
FROM datagen_complex_source;
```

- **示例5：模拟高负载场景，控制数据量和速率进行性能压测。**

```
-- 创建DataGen源表（高吞吐量压测配置）
CREATE TABLE datagen_pressure_source (
  id BIGINT,
  payload STRING,
  event_time TIMESTAMP(3),
  WATERMARK FOR event_time AS event_time - INTERVAL '1' SECOND
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100000',
  'number-of-rows' = '10000000',
  'fields.id.kind' = 'sequence',
  'fields.id.start' = '1',
  'fields.id.end' = '10000000',
  'fields.payload.length' = '1024'
);

-- 创建BlackHole结果表
CREATE TABLE blackhole_sink (
  id BIGINT,
  payload STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'blackhole'
);

-- 提交作业：测试纯处理吞吐量
INSERT INTO blackhole_sink
SELECT id, payload, event_time
FROM datagen_pressure_source;
```

1.7 BlackHole 连接器

概述

BlackHole连接器是一个黑洞Sink，丢弃所有写入的数据，不产生任何输出。它类似于Unix系统中的/dev/null，用于不需要实际输出的场景。

 版权所有 © 华为云计算技术有限公司

表 1-6 连接器详情

类别	详情
支持类型	结果表
运行模式	批模式和流模式
数据格式	暂不适用
API种类	SQL
是否支持更新或删除结果表数据	是

适用场景

- **性能测试**：测试Flink SQL作业的Source端和计算层吞吐量，排除Sink端I/O影响。
- **逻辑验证**：验证SQL语法和作业DAG是否正确构建，无需关心输出结果。
- **开发调试**：在仅需观察Source或中间处理结果的场景下，作为最终Sink占位。

约束与限制

- 仅支持作为Sink使用，不支持作为Source。
- 所有写入数据将被丢弃，无法恢复。
- 不适用于生产环境的数据持久化场景。

语法结构

```
CREATE TABLE {table_name} (
  {column_name} {column_type},
  ...
) WITH (
  'connector' = 'blackhole'
);
```

WITH 参数说明

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	blackhole	标识连接器类型为BlackHole连接器。 必须填写且值固定为blackhole。Flink运行时根据此值加载对应的BlackHole SinkFactory，所有写入数据在运行时被直接丢弃，不产生任何I/O操作。

说明

BlackHole连接器除connector参数外，无需配置任何其他参数。

版权所有 © 华为云计算技术有限公司

使用示例

- **示例1：创建一个BlackHole Sink表，丢弃所有写入数据。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  id BIGINT,
  name STRING,
  `value` DOUBLE
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1000',
  'fields.id.kind' = 'sequence',
  'fields.id.start' = '1',
  'fields.id.end' = '100000'
);

-- 创建BlackHole结果表
CREATE TABLE blackhole_sink (
  id BIGINT,
  name STRING,
  `value` DOUBLE
) WITH (
  'connector' = 'blackhole'
);

-- 提交作业
INSERT INTO blackhole_sink
SELECT id, name, `value`
FROM datagen_source;
```

- **示例2：使用DataGen生成高吞吐量数据，通过BlackHole Sink接收，测试Source端和计算层的纯处理能力。**

```
-- 创建DataGen源表（高吞吐量配置）
CREATE TABLE datagen_source (
  id BIGINT,
  payload STRING,
  event_time TIMESTAMP(3),
  WATERMARK FOR event_time AS event_time - INTERVAL '1' SECOND
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '100000',
  'fields.id.kind' = 'sequence',
  'fields.id.start' = '1',
  'fields.id.end' = '10000000',
  'fields.payload.length' = '100'
);

-- 创建BlackHole结果表
CREATE TABLE blackhole_sink (
  id BIGINT,
  payload STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'blackhole'
);

-- 提交作业：测试纯处理吞吐量
INSERT INTO blackhole_sink
SELECT id, payload, event_time
FROM datagen_source;
```

- **示例3：作业需要将数据分流处理，部分数据写入Kafka，另一部分数据暂不需要输出，使用BlackHole作为占位。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  user_id INT,
  event_type STRING,
  event_time TIMESTAMP(3)
) WITH (
```

 版权所有 © 华为云计算技术有限公司

```
  'connector' = 'datagen',
  'rows-per-second' = '100',
  'fields.user_id.kind' = 'sequence',
  'fields.user_id.start' = '1',
  'fields.user_id.end' = '100000'
);

-- 主输出：写入Kafka
CREATE TABLE kafka_sink (
  user_id INT,
  event_type STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'kafka',
  'topic' = 'user-events',
  'properties.bootstrap.servers' = 'kafka-broker:9092',
  'format' = 'json'
);

-- 占位输出：丢弃不需要的数据
CREATE TABLE blackhole_sink (
  user_id INT,
  event_type STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'blackhole'
);

-- 分流处理：登录事件写入Kafka，其他事件丢弃
EXECUTE STATEMENT SET
BEGIN
INSERT INTO kafka_sink
SELECT user_id, event_type, event_time
FROM datagen_source
WHERE event_type = 'login';

INSERT INTO blackhole_sink
SELECT user_id, event_type, event_time
FROM datagen_source
WHERE event_type <> 'login';
END;
```

1.8 Print 连接器

概述

Print连接器将数据输出到TaskManager的标准输出（日志），常用于开发和调试阶段快速查看Flink SQL作业的输出结果。

表 1-7 连接器详情

类别	详情
支持类型	结果表
运行模式	批模式和流模式
数据格式	暂不适用
API种类	SQL

 版权所有 © 华为云计算技术有限公司

类别	详情
是否支持更新或删除结果表数据	是

适用场景

- **开发调试**：快速验证Flink SQL处理逻辑的输出结果，无需搭建外部存储。
- **问题排查**：在作业运行过程中实时观察数据流内容，定位数据处理问题。
- **数据抽样**：通过标准输出查看少量数据样本，确认数据格式和内容。

约束与限制

- 仅支持作为Sink使用，不支持作为Source。
- 大数据量场景下可能影响性能，不建议在生产环境中使用。

语法结构

```
CREATE TABLE {table_name} (
  {column_name} {column_type},
  ...
) WITH (
  'connector' = 'print'
);
```

WITH 参数说明

参数	是否必填	类型	默认值	取值范围	说明
connector	是	String	无	print	标识连接器类型为Print连接器。 必须填写且值固定为print。Flink运行时根据此值加载对应的Print SinkFactory，将表声明为Print输出表。

使用示例

- **示例1：创建一个最简Print Sink表，将数据直接输出到TaskManager日志。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  id BIGINT,
  name STRING,
  score DOUBLE
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '10',
  'fields.id.kind' = 'sequence',
  'fields.id.start' = '1',
  'fields.id.end' = '100'
);

-- 创建Print结果表
```

 版权所有 © 华为云计算技术有限公司

```
CREATE TABLE print_sink (
  id BIGINT,
  name STRING,
  score DOUBLE
) WITH (
  'connector' = 'print'
);

-- 提交作业
INSERT INTO print_sink
SELECT id, name, score
FROM datagen_source;
```

- **示例2：当作业中存在多个Print Sink时，通过标识前缀区分不同数据流的输出。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  user_id INT,
  event_type STRING,
  event_time TIMESTAMP(3),
  WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '10',
  'fields.user_id.kind' = 'sequence',
  'fields.user_id.start' = '1',
  'fields.user_id.end' = '100'
);

-- 创建Print结果表（带标识前缀）
CREATE TABLE print_sink_with_id (
  user_id INT,
  event_type STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'print',
  'print-identifier' = 'user-event'
);

-- 提交作业
INSERT INTO print_sink_with_id
SELECT user_id, event_type, event_time
FROM datagen_source;
```

输出效果示例：

```
user-event> +I[1, login, 2024-01-01 10:00:00.000]
user-event> +I[2, click, 2024-01-01 10:01:00.000]
```

- **示例3：使用DataGen生成模拟数据，通过Print输出结果，构建完整的零依赖测试链路。**

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
  id INT,
  name STRING,
  event_time TIMESTAMP(3),
  WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '10',
  'fields.id.kind' = 'sequence',
  'fields.id.start' = '1',
  'fields.id.end' = '100'
);

-- 创建Print结果表
CREATE TABLE print_sink (
  id INT,
  name STRING,
  event_time TIMESTAMP(3)
) WITH (
  'connector' = 'print',
```

 版权所有 © 华为云计算技术有限公司

```
  'print-identifier' = 'debug-output'
);

-- 提交作业：将源表数据写入结果表
INSERT INTO print_sink
SELECT id, name, event_time
FROM datagen_source;
```

 版权所有 © 华为云计算技术有限公司

2 关键字

Flink V1.20关键字详情，请参见**Reserved Keywords**。

说明

如果您需要在DDL中定义Flink关键字，请在关键字两端添加反引号（`），例如`value`。

版权所有 © 华为云计算技术有限公司

3 DDL 语句

3.1 CREATE CATALOG

概述

CREATE CATALOG语句用于在当前会话中注册外部Catalog。Catalog是Flink元数据管理的顶层命名空间，包含数据库、表、视图和函数等对象。AI DataLake当前确认支持HiveCatalog，用于连接Hive Metastore实现Flink与Hive的数据互操作。

适用场景

- **Hive数据集成**：通过HiveCatalog访问Hive中的表数据，在Flink SQL中进行流批一体化处理。
- **元数据统一管理**：将Hive元数据纳入Flink的Catalog体系，实现统一查询与管理。
- **跨数据源查询**：注册多个Catalog后，在SQL中通过全限定名（catalog.database.table）跨数据源查询。

约束与限制

- 使用HiveCatalog需确保Hive Metastore服务可用且网络连通。
- Catalog注册后仅在当前会话有效，不会持久化到集群级别。
- 注册同名Catalog会覆盖已有Catalog，需使用IF NOT EXISTS避免冲突。

语法结构

```
CREATE CATALOG [IF NOT EXISTS] catalog_name
WITH (key1=val1, key2=val2, ...)
```

使用示例

示例1：创建HiveCatalog

```
-- 创建HiveCatalog
CREATE CATALOG IF NOT EXISTS myhive WITH
WITH (
    'type' = 'hive',
    'metastore' = 'lakeformation'
```

 版权所有 © 华为云计算技术有限公司

```
 );
-- 创建DataGen源表
CREATE TABLE datagen_source (
id BIGINT,
name STRING,
`value` DOUBLE
) WITH (
'connector' = 'datagen',
'rows-per-second' = '100',
'fields.id.kind' = 'sequence',
'fields.id.start' = '1',
'fields.id.end' = '10000',
'fields.name.length' = '8',
'fields.value.min' = '0.0',
'fields.value.max' = '100.0'
);
-- 创建Print结果表
CREATE TABLE print_sink (
id BIGINT,
name STRING,
`value` DOUBLE
) WITH (
'connector' = 'print',
'print-identifier' = 'catalog_output'
);
-- 提交作业
INSERT INTO print_sink
SELECT id, name, `value`
FROM datagen_source;
```

3.2 CREATE DATABASE

概述

CREATE DATABASE语句用于在当前或指定Catalog中创建数据库。数据库是Catalog下的命名空间，用于组织和管理表、视图、函数等对象。创建数据库后可通过USE语句切换到该数据库进行后续操作。

适用场景

- **项目隔离**：为不同项目或业务域创建独立数据库，实现数据逻辑隔离。
- **环境管理**：为开发、测试、生产等不同环境分别创建数据库。
- **权限分区**：通过数据库级别划分访问权限，控制不同用户的数据可见范围。

约束与限制

- 数据库名在同一个Catalog内必须唯一，重复创建需使用IF NOT EXISTS。
- WITH子句中可设置的属性取决于Catalog实现，不同Catalog支持的属性不同。
- 数据库只能在当前Catalog下创建，不支持跨Catalog创建。
- 删除数据库前需确保数据库内无表或其他对象，或使用CASCADE级联删除。

语法结构

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

 版权所有 © 华为云计算技术有限公司

语法说明

参数	必填	说明
IF NOT EXISTS	否	指定当同名数据库已存在时不报错。未指定时若同名数据库存在则抛出异常。
catalog_name	否	指定目标Catalog名称。省略时使用当前Catalog。
db_name	是	数据库名称，在Catalog内必须唯一。命名建议使用小写字母和下划线。
COMMENT	否	数据库的注释说明，便于理解数据库用途和所属业务。
WITH	否	以键值对形式设置数据库级别属性。具体可用属性取决于Catalog实现。

使用示例

示例1：创建数据库

```
CREATE DATABASE IF NOT EXISTS mydb COMMENT 'AI DataLake业务数据库';
```

示例2：带WITH属性创建paimon数据库

```
CREATE DATABASE IF NOT EXISTS paimon_db
COMMENT 'paimon元数据数据库'
WITH (
 'location' = 'obs://bucket/path/paimon-warehouse/paimon_db'
);
```

3.3 CREATE TABLE

概述

CREATE TABLE语句用于在当前或指定Catalog中注册表，定义表的结构（物理列、计算列、元数据列）、时间属性（Watermark）、约束（主键）和连接器配置（WITH子句）。注册后的表可在SQL查询中作为源表或结果表使用。AI DataLake支持Flink SQL标准建表语法，涵盖分区表、LIKE建表等核心能力。

适用场景

- **数据接入**：通过CREATE TABLE定义外部数据源（如Kafka、HDFS）的映射表，实现数据读取与写入。
- **流式ETL**：定义带Watermark和计算列的源表，配合窗口函数完成实时数据转换。
- **分区管理**：使用PARTITIONED BY创建分区表，优化大数据量场景下的查询性能。
- **表结构复用**：通过LIKE子句基于已有表快速创建新表，减少重复定义。

约束与限制

- AI DataLake不支持CREATE TEMPORARY TABLE语法，所有表均为持久化表。

版权所有 © 华为云计算技术有限公司

- 主键约束声明为NOT ENFORCED，Flink不会校验数据唯一性，需依赖外部存储保证。
- 一个表只能定义一个Watermark，即只能声明一个事件时间字段。
- PARTITIONED BY仅支持在连接器支持分区的场景下使用（如Hive、Paimon等），Kafka等连接器不支持分区表。
- 计算列不能引用其他计算列，只能引用物理列和元数据列。
- 元数据列的可用性取决于连接器实现，不同连接器支持的元数据不同。
- LIKE子句的INCLUDING/EXCLUDING选项需与源表结构匹配，错误的组合会导致建表失败。

语法结构

完整语法

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

列定义语法

```
-- 物理列
<physical_column_definition>:
  column_name column_type [ <column_constraint> ] [COMMENT column_comment]

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

-- 元数据列
<metadata_column_definition>:
  column_name column_type METADATA [ FROM metadata_key ] [ VIRTUAL ]

-- 计算列
<computed_column_definition>:
  column_name AS computed_column_expression [COMMENT column_comment]
```

Watermark定义语法

```
<watermark_definition>:
  WATERMARK FOR rowtime_column_name AS watermark_strategy_expression
```

主键约束语法

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

LIKE选项语法

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


版权所有 © 华为云计算技术有限公司

参数说明

主句参数

参数	必填	说明
IF NOT EXISTS	否	指定当同名表已存在时不报错，直接跳过创建。未指定时若同名表存在则抛出异常。
catalog_name.db_name	否	指定表所在的Catalog和数据库名。未指定时使用当前Catalog和数据库。
table_name	是	表名，在同一数据库内必须唯一。
COMMENT table_comment	否	为表添加注释说明。
PARTITIONED BY	否	指定分区列，将表按分区列值存储到不同目录。仅连接器支持分区时有效，分区列必须是表中已定义的物理列。
WITH (key1=val1, ...)	是	指定表的连接器配置项，以键值对形式提供。connector参数为必填项，决定数据来源或去向。
LIKE source_table	否	基于已有表的结构创建新表，支持INCLUDING/EXCLUDING/OVERWRITING控制结构继承策略。可复用表结构而避免重复定义。

列类型参数

列类型	关键字	说明
物理列	无特殊关键字	定义物理介质中存储数据的字段，是最常用的列定义方式。物理列的顺序与数据源中的字段顺序对应。
计算列	AS	通过表达式生成的虚拟列，不物理存储在表中。常用于定义Processing Time（AS PROCTIME()）或派生字段。只能引用物理列和元数据列，不能引用其他计算列。
元数据列	METADATA	访问连接器数据源的元数据信息，如Kafka的timestamp、offset等。VIRTUAL标记表示该元数据仅读取不持久化。可用性取决于连接器实现。

Watermark参数

版权所有 © 华为云计算技术有限公司

参数	必填	说明
rowtime_column_name	是	指定事件时间列，必须为TIMESTAMP(3)或TIMESTAMP_LTZ(3)类型。一个表只能定义一个Watermark。
watermark_strategy_expression	是	水位线生成策略表达式，用于计算Watermark的时间戳。常用策略为rowtime_column - INTERVAL 'n' SECOND，表示允许最大n秒的乱序。

主键约束参数

参数	必填	说明
constraint_name	否	约束名称，可选。
column_name, ...	是	主键包含的一个或多个列，这些列不允许为NULL。
NOT ENFORCED	是	Flink不强制校验主键唯一性，需依赖外部存储保证。此关键字为必填，不可省略。主键用于upsert连接器的语义判断。

LIKE选项参数

选项	可选策略	说明
ALL	INCLUDING / EXCLUDING	控制是否包含源表的所有特性。INCLUDING ALL表示全部继承，EXCLUDING ALL表示全部排除。
CONSTRAINTS	INCLUDING / EXCLUDING	控制是否包含主键等约束。INCLUDING CONSTRAINTS继承约束，EXCLUDING CONSTRAINTS不继承。
PARTITIONS	INCLUDING / EXCLUDING	控制是否包含分区信息。INCLUDING PARTITIONS继承分区定义，EXCLUDING PARTITIONS不继承。
GENERATED	INCLUDING / EXCLUDING / OVERWRITING	控制计算列和元数据列的继承。INCLUDING继承，EXCLUDING排除，OVERWRITING用新定义覆盖源表计算列。
OPTIONS	INCLUDING / EXCLUDING / OVERWRITING	控制WITH选项的继承。INCLUDING OPTIONS继承所有连接器配置，EXCLUDING OPTIONS不继承，OVERWRITING OPTIONS用新配置覆盖同名项。

版权所有 © 华为云计算技术有限公司

选项	可选策略	说明
WATERMARKS	INCLUDING / EXCLUDING / OVERWRITING	控制Watermark定义的继承。INCLUDING WATERMARKS继承水位线，EXCLUDING WATERMARKS不继承，OVERWRITING WATERMARKS用新定义覆盖。

使用示例

AI DataLake支持多种连接器，各连接器具有特定的使用示例，详情请参见**1.1 连接器的支持情况**查看各连接器文档中的使用示例。

3.4 CREATE VIEW

概述

CREATE VIEW语句用于基于SELECT查询创建虚拟表。视图不存储数据，每次访问时动态执行底层查询。视图可用于简化复杂查询、提供逻辑数据抽象层和实现访问控制。

适用场景

- **查询简化**：将多表JOIN、聚合等复杂查询封装为视图，在后续SQL中直接引用视图名。
- **数据脱敏**：创建仅包含可公开字段的视图，隐藏敏感列。
- **逻辑分层**：在ETL流程中通过视图定义中间处理逻辑，提高SQL可读性和可维护性。

约束与限制

- AI DataLake不支持CREATE TEMPORARY VIEW语法。
- 视图是只读的，不支持通过视图插入或更新数据。
- 视图定义中引用的表或视图必须已存在，否则创建失败。
- 视图不支持PARTITIONED BY、WITH等表属性子句。
- 当底层表结构发生变更（如删除列）时，视图可能失效。

语法结构

```
CREATE VIEW [IF NOT EXISTS] [catalog_name.][db_name.]view_name
[( columnName [, columnName ]* )]
[COMMENT view_comment]
AS query_expression
```

语法说明

参数	必填	说明
IF NOT EXISTS	否	指定当同名视图已存在时不报错。未指定时若同名视图存在则抛出异常。

版权所有 © 华为云计算技术有限公司

参数	必填	说明
view_name	是	视图名称，在数据库内必须唯一。可指定catalog_name和db_name前缀。
columnName	否	自定义视图的列名列表。省略时使用查询结果中的列名。列名数量须与查询结果列数一致。
COMMENT	否	视图的注释说明，便于理解视图用途和业务含义。
query_expression	是	定义视图内容的SELECT查询语句。查询中可引用表、其他视图和函数。

使用示例

示例1：创建视图

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
user_id BIGINT,
product_id BIGINT,
amount DECIMAL(10, 2),
category STRING,
ts TIMESTAMP(3),
WATERMARK FOR ts AS ts - INTERVAL '5' SECOND
) WITH (
'connector' = 'datagen',
'rows-per-second' = '100',
'fields.user_id.kind' = 'sequence',
'fields.user_id.start' = '1',
'fields.user_id.end' = '10000',
'fields.product_id.kind' = 'sequence',
'fields.product_id.start' = '100',
'fields.product_id.end' = '200',
'fields.category.length' = '6'
);
-- 创建Print结果表
CREATE TABLE print_sink (
user_id BIGINT,
total_amount DECIMAL(10, 2)
) WITH (
'connector' = 'print',
'print-identifier' = 'view_output'
);
-- 创建视图：按用户汇总金额
CREATE VIEW IF NOT EXISTS user_amount_view AS
SELECT user_id, SUM(amount) AS total_amount
FROM datagen_source
GROUP BY user_id;
-- 提交作业
INSERT INTO print_sink
SELECT user_id, total_amount
FROM user_amount_view;
```

3.5 CREATE FUNCTION

概述

CREATE FUNCTION语句用于在当前Catalog中注册用户自定义函数（UDF），包括标量函数、表函数和聚合函数。注册后的函数可在SQL查询中像内置函数一样使用。

版权所有 © 华为云计算技术有限公司

适用场景

- **业务逻辑封装**：将复杂的业务计算逻辑封装为UDF，在SQL中简洁调用。
- **格式转换**：自定义数据格式解析函数，处理非标准JSON、XML等数据。
- **加密脱敏**：实现自定义加密、脱敏算法，满足数据安全要求。

约束与限制

- AI DataLake不支持TEMPORARY关键字，无法创建临时函数。
- 自定义函数的JAR包需提前上传至集群，确保类路径可访问。
- 函数类必须实现Flink对应的UDF接口（如ScalarFunction、TableFunction、AggregateFunction）。
- 函数名在同一个数据库内必须唯一。

语法结构

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

语法说明

参数	必填	说明
SYSTEM	否	声明为系统函数。系统函数的解析优先级高于内置函数，可覆盖同名内置函数的行为。
IF NOT EXISTS	否	指定当同名函数已存在时不报错。未指定时若同名函数存在则抛出异常。
function_name	是	函数名称，在数据库内必须唯一。可指定catalog_name和db_name前缀。
class_identifier	是	UDF实现类的全限定类名。该类必须实现Flink对应的UDF接口，且JAR包已在集群中可用。
LANGUAGE	否	指定UDF的实现语言，支持JAVA、SCALA、PYTHON。默认为JAVA。Python UDF需额外配置Python环境。

使用示例

示例：创建Java自定义函数

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
id BIGINT,
raw_text STRING
) WITH (
'connector' = 'datagen',
'rows-per-second' = '10',
'fields.id.kind' = 'sequence',
'fields.id.start' = '1',
'fields.id.end' = '1000',
'fields.raw_text.length' = '10'
);
```


版权所有 © 华为云计算技术有限公司

```
-- 创建Print结果表
CREATE TABLE print_sink (
id BIGINT,
processed_text STRING
) WITH (
'connector' = 'print'
);
-- 注册自定义函数
CREATE FUNCTION IF NOT EXISTS my_udf AS 'com.example.MyScalarFunction';
-- 提交作业
INSERT INTO print_sink
SELECT id, my_udf(raw_text) AS processed_text
FROM datagen_source;
```

3.6 ALTER 语句

概述

ALTER语句用于修改已注册到Catalog中的对象定义，包括表、视图、数据库、函数和Catalog。通过ALTER语句可在不重建对象的情况下完成Schema演进、属性修改、重命名等操作。

适用场景

- **Schema演进**：业务需求变更时通过ALTER TABLE ADD/DROP COLUMN在线调整表结构，无需重建表。
- **参数调优**：运行中需要调整连接器配置参数，通过ALTER TABLE SET修改WITH属性。
- **对象管理**：对表或视图进行重命名，修正命名不规范问题。

约束与限制

- ALTER TABLE添加的列默认追加到表末尾，不支持指定列的位置。
- 修改Watermark或主键约束时，需确保与现有列定义兼容。
- ALTER DATABASE修改的属性取决于Catalog实现，不同Catalog支持不同的属性。
- ALTER VIEW仅支持重命名和修改属性，不支持修改视图的查询定义。
- 对正在被作业使用的表执行ALTER操作可能导致作业异常，建议在作业停止后执行。
- ALTER TABLE仅修改元数据，是否可以修改物理表结构取决于连接器实现。

语法结构

ALTER TABLE

-- 重命名表

```
ALTER TABLE [catalog_name.][db_name.]table_name RENAME TO new_table_name;
```

-- 修改表属性

```
ALTER TABLE [catalog_name.][db_name.]table_name SET (key1=val1, key2=val2, ...);
```

-- 添加列

 版权所有 © 华为云计算技术有限公司

```
ALTER TABLE [catalog_name.][db_name.]table_name ADD (col_name col_type [COMMENT
col_comment], ...);
```

-- 删除列

```
ALTER TABLE [catalog_name.][db_name.]table_name DROP (col_name, ...);
```

-- 添加Watermark

```
ALTER TABLE [catalog_name.][db_name.]table_name ADD WATERMARK FOR rowtime_col AS
watermark_expr;
```

-- 删除Watermark

```
ALTER TABLE [catalog_name.][db_name.]table_name DROP WATERMARK;
```

-- 添加主键

```
ALTER TABLE [catalog_name.][db_name.]table_name ADD PRIMARY KEY (col_name, ...) NOT ENFORCED;
```

-- 删除主键

```
ALTER TABLE [catalog_name.][db_name.]table_name DROP PRIMARY KEY;
```

-- 修改列名

```
ALTER TABLE [catalog_name.][db_name.]table_name RENAME col_name TO new_col_name;
```

-- 修改列类型

```
ALTER TABLE [catalog_name.][db_name.]table_name MODIFY (col_name new_col_type [COMMENT
col_comment], ...);
```

ALTER VIEW

-- 重命名视图

```
ALTER VIEW [catalog_name.][db_name.]view_name RENAME TO new_view_name;
```

-- 修改视图属性

```
ALTER VIEW [catalog_name.][db_name.]view_name SET (key1=val1, key2=val2, ...);
```

ALTER DATABASE

-- 修改数据库属性

```
ALTER DATABASE [catalog_name.]db_name SET (key1=val1, key2=val2, ...);
```

ALTER FUNCTION

-- 修改函数定义

```
ALTER [SYSTEM] FUNCTION [IF EXISTS] [catalog_name.][db_name.]function_name
AS class_identifier
[LANGUAGE JAVA|SCALA|PYTHON];
```

ALTER CATALOG

-- 修改Catalog属性

```
ALTER CATALOG catalog_name SET (key1=val1, key2=val2, ...);
```

使用示例

示例：ALTER TABLE添加列和修改属性

```
-- ALTER TABLE paimon_append_source ADD (email STRING COMMENT '用户邮箱');
-- ALTER TABLE paimon_append_source SET ('rows-per-second' = '200');
```

 版权所有 © 华为云计算技术有限公司

3.7 DROP 语句

概述

DROP语句用于从Catalog中删除已注册的对象，包括表、数据库、视图、函数和Catalog。删除操作会移除对象的元数据定义，对于部分连接器（如Hive）还会删除底层的物理数据。

适用场景

- **资源清理**：删除不再使用的表、视图或数据库，释放元数据和存储资源。
- **环境重置**：在开发或测试过程中删除临时对象，恢复到初始状态。
- **函数管理**：注销不再使用的自定义函数，避免命名冲突。

约束与限制

- DROP TABLE仅删除元数据，是否删除物理数据取决于连接器实现。HiveCatalog会同时删除物理数据，其他连接器可能只删除元数据。
- DROP DATABASE默认使用RESTRICT模式，数据库非空时抛出异常。使用CASCADE可级联删除数据库中的所有对象。
- 删除正在被作业使用的表或视图可能导致作业异常，建议在作业停止后执行。
- DROP CATALOG仅移除会话中的Catalog注册信息，不影响外部Metastore中的数据。
- DROP FUNCTION删除函数后，引用该函数的SQL查询将无法执行。

语法结构

DROP TABLE

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

DROP DATABASE

```
DROP DATABASE [IF EXISTS] [catalog_name.]db_name [ (RESTRICT | CASCADE) ];
```

DROP VIEW

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

DROP FUNCTION

```
DROP [SYSTEM] FUNCTION [IF EXISTS] [catalog_name.][db_name.]function_name;
```

DROP CATALOG

```
DROP CATALOG [IF EXISTS] catalog_name;
```

语法说明

参数	必填	说明
IF EXISTS	否	指定当对象不存在时不报错。未指定时若对象不存在则抛出异常。建议始终使用以避免脚本中断。

版权所有 © 华为云计算技术有限公司

参数	必填	说明
RESTRICT	否	DROP DATABASE的默认模式。数据库非空时抛出异常，阻止误删含对象的数据库。
CASCADE	否	级联删除数据库中的所有表、视图和函数。使用时需谨慎，确保确认删除范围。
SYSTEM	否	标识删除的是系统函数。系统函数的解析优先级高于内置函数，删除后恢复内置函数的默认行为。

使用示例

示例：DROP TABLE和DROP DATABASE

```
-- DROP TABLE IF EXISTS my_table;
-- DROP DATABASE IF EXISTS my_db CASCADE;
-- DROP VIEW IF EXISTS my_view;
-- DROP FUNCTION IF EXISTS my_udf;
```


版权所有 © 华为云计算技术有限公司

4 DML 语句

4.1 INSERT INTO

概述

INSERT INTO语句是Flink SQL中核心的数据写入语句，用于将SELECT查询的结果或显式指定的值写入目标表（Sink表）。AI DataLake支持INSERT INTO，即向目标表追加数据，不覆盖已有数据。这是最常用的写入模式，适用于流式和批处理场景。

此外，AI DataLake支持通过STATEMENT SET语法将多条INSERT语句合并为一个Flink作业执行，使多条写入操作共享同一个DAG（有向无环图），从而降低TaskManager资源开销，适用于多Sink分流场景。

适用场景

- **实时数据入湖**：将Kafka等消息队列中的实时数据通过INSERT INTO写入Paimon或Hive等存储，构建实时数仓。
- **多Sink分流**：同一数据源经ETL处理后需要写入多个目标表（如明细表、聚合表同时写入），使用STATEMENT SET合并为一个作业，减少资源消耗。
- **动态参数调优**：通过Table Hints在INSERT语句中动态调整连接器参数，无需修改表DDL即可适应不同运行环境。

约束与限制

- STATEMENT SET中的多条INSERT语句作为一个整体提交，无法单独控制某一条INSERT的启停。
- INSERT语句中SELECT子句的列数量和类型必须与目标表的列定义匹配，或通过显式column_list指定对应列。
- 分区写入时，动态分区列必须出现在SELECT子句的最后，且与PARTITION子句中声明的顺序一致。
- Table Hints（/*+ OPTIONS(...) */）仅对当前INSERT语句中的表实例生效，不会修改表在Catalog中的持久化定义。

语法结构

INSERT INTO单Sink写入

 版权所有 © 华为云计算技术有限公司

```
INSERT INTO [catalog_name.][db_name.]table_name
[PARTITION part_spec]
[column_list]
select_statement | VALUES (value [, value ...]) [, (value [, value ...]) ...];
part_spec ::= (key1=value1, key2=value2, ...)
column_list ::= (col1 [, col2, ...])
```

INSERT INTO多Sink写入

```
EXECUTE STATEMENT SET
BEGIN
INSERT INTO sink_table1 select_statement1;
INSERT INTO sink_table2 select_statement2;
[...]
END;
```

说明

EXECUTE STATEMENT SET与END之间的所有INSERT语句将被合并为一个Flink作业。多条INSERT共享Source算子，合并到同一DAG中执行，有效减少TaskManager资源占用。

Table Hints动态传参

```
INSERT INTO table_name /*+ OPTIONS('key1'='value1', 'key2'='value2') */
select_statement;
```

使用示例

示例1：INSERT INTO单Sink写入

使用DataGen连接器生成模拟传感器数据，通过INSERT INTO写入Print连接器打印输出。

```
-- 创建传感器数据源表
CREATE TABLE sensor_source (
sensor_id INT,
temperature DOUBLE,
humidity DOUBLE,
record_time TIMESTAMP(3),
WATERMARK FOR record_time AS record_time - INTERVAL '5' SECOND
) WITH (
'connector' = 'datagen',
'rows-per-second' = '10',
'fields.sensor_id.kind' = 'random',
'fields.sensor_id.min' = '1',
'fields.sensor_id.max' = '100',
'fields.temperature.min' = '15.0',
'fields.temperature.max' = '40.0',
'fields.humidity.min' = '30.0',
'fields.humidity.max' = '90.0'
);
-- 创建传感器数据结果表
CREATE TABLE sensor_sink (
sensor_id INT,
temperature DOUBLE,
humidity DOUBLE
) WITH (
'connector' = 'print',
'print-identifier' = 'sensor_data'
);
-- 提交作业：筛选温度超过30度的传感器数据
INSERT INTO sensor_sink
SELECT sensor_id, temperature, humidity
FROM sensor_source
WHERE temperature > 30;
```

示例2：INSERT INTO多Sink写入（STATEMENT SET）

 版权所有 © 华为云计算技术有限公司

使用DataGen连接器生成订单数据，通过STATEMENT SET将数据分别写入明细表和聚合统计表，两条INSERT在同一个Flink作业中执行。

-- 创建订单数据源表

```
CREATE TABLE order_source (
order_id BIGINT,
customer_id BIGINT,
product_id BIGINT,
amount DECIMAL(10, 2),
order_time TIMESTAMP(3),
WATERMARK FOR order_time AS order_time - INTERVAL '5' SECOND
) WITH (
'connector' = 'datagen',
'rows-per-second' = '100',
'fields.order_id.kind' = 'sequence',
'fields.order_id.start' = '1',
'fields.order_id.end' = '1000000',
'fields.customer_id.kind' = 'random',
'fields.customer_id.min' = '1',
'fields.customer_id.max' = '5000',
'fields.product_id.kind' = 'random',
'fields.product_id.min' = '1',
'fields.product_id.max' = '200',
'fields.amount.min' = '10.00',
'fields.amount.max' = '1000.00'
);
-- 创建订单明细结果表
CREATE TABLE order_detail_sink (
order_id BIGINT,
customer_id BIGINT,
amount DECIMAL(10, 2)
) WITH (
'connector' = 'print',
'print-identifier' = 'order_detail'
);
-- 创建客户聚合结果表
CREATE TABLE customer_agg_sink (
customer_id BIGINT,
order_count BIGINT,
total_amount DECIMAL(10, 2)
) WITH (
'connector' = 'print',
'print-identifier' = 'customer_agg'
);
-- 提交作业：多Sink写入，两条INSERT共享同一DAG
EXECUTE STATEMENT SET
BEGIN
INSERT INTO order_detail_sink
  SELECT order_id, customer_id, amount
FROM order_source;
INSERT INTO customer_agg_sink
  SELECT customer_id, COUNT(*) AS order_count, SUM(amount) AS total_amount
FROM order_source
GROUP BY customer_id;
END;
```


版权所有 © 华为云计算技术有限公司

5 Queries 语句

Queries语句	相关文档
Hints	SQL Hints
WITH子句	WITH clause
SELECT与WHERE子句	SELECT & WHERE clause
SELECT DISTINCT	SELECT DISTINCT
窗口函数	Windowing table-valued functions (Windowing TVFs)
窗口聚合	Window Aggregation
分组聚合	Group Aggregation
Over聚合	Over Aggregation
Join	Joins
窗口关联	Window Join
集合操作	Set Operations
ORDER BY语句	ORDER BY clause
LIMIT语句	LIMIT clause
Top-N	Top-N
窗口Top-N	Window Top-N
去重	Deduplication
窗口去重	Window Deduplication
模式检测	Pattern Recognition

版权所有 © 华为云计算技术有限公司

Queries语句	相关文档
时间旅行	**Time Travel** **说明** 当前仅Paimon Catalog支持时间旅行，详情请参见**Time Travel**。

版权所有 © 华为云计算技术有限公司

6 内置函数

6.1 比较函数

概述

比较函数用于对两个表达式进行比较运算，返回BOOLEAN类型结果。AI DataLake支持SQL标准中的等值比较、大小比较、范围比较、模式匹配和空值判断等完整的比较运算能力。

更多内容请参考**Comparison Functions**。

适用场景

- **WHERE过滤**：在查询中使用比较条件过滤数据行。
- **JOIN关联**：在JOIN条件中使用等值比较关联不同表。
- **条件分支**：在CASE WHEN中使用比较结果控制业务逻辑。
- **数据校验**：通过比较运算检测数据的有效性和一致性。

约束与限制

- 比较运算的两端数据类型必须兼容，否则需显式CAST转换。
- NULL与任何值比较结果均为NULL（不是TRUE也不是FALSE），需使用IS NULL判断。
- BETWEEN 是闭区间，包含两端边界值，即value1 BETWEEN value2 AND value3等价于value1 >= value2 AND value1 <= value3。
- LIKE中%匹配任意字符序列（包括空序列），_匹配恰好一个字符；如需匹配通配符本身，需使用ESCAPE子句转义。

  若IN列表中包含NULL，且value1不在列表的非NULL值中，则结果为NULL而非FALSE。

版权所有 © 华为云计算技术有限公司

语法结构

语法	返回类型	说明	示例
value1 = value2	BOOLEAN	等值比较。 若value1等于value2则返回TRUE，否则返回FALSE；value1和value2类型必须兼容；若任一操作数为NULL则返回NULL。	SELECT 1 = 1; -- TRUE
value1 <> value2	BOOLEAN	不等比较。 若value1不等于value2则返回TRUE，否则返回FALSE；value1和value2类型必须兼容；若任一操作数为NULL则返回NULL。	SELECT 1 <> 2; -- TRUE
value1 != value2	BOOLEAN	不等比较。 与 <> 语义完全等价，若value1不等于value2则返回TRUE；若任一操作数为NULL则返回NULL。	SELECT 1 != 2; -- TRUE
value1 > value2	BOOLEAN	大于比较。 若value1大于value2则返回TRUE，否则返回FALSE；value1和value2类型必须兼容且可排序；若任一操作数为NULL则返回NULL。	SELECT 3 > 2; -- TRUE
value1 >= value2	BOOLEAN	大于等于比较。 若value1大于或等于value2则返回TRUE，否则返回FALSE；value1和value2类型必须兼容且可排序；若任一操作数为NULL则返回NULL。	SELECT 2 >= 2; -- TRUE
value1 < value2	BOOLEAN	小于比较。 若value1小于value2则返回TRUE，否则返回FALSE；value1和value2类型必须兼容且可排序；若任一操作数为NULL则返回NULL。	SELECT 1 < 2; -- TRUE
value1 <= value2	BOOLEAN	小于等于比较。 若value1小于或等于value2则返回TRUE，否则返回FALSE；value1和value2类型必须兼容且可排序；若任一操作数为 NULL则返回NULL。	SELECT 2 <= 3; -- TRUE

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
value IS NULL	BOOLEAN	空值判断。 若value为NULL则返回TRUE，否则返回FALSE；不受NULL 参与常规比较返回NULL的规则影响，是判断NULL值的标准方式。	SELECT NULL IS NULL; -- TRUE
value IS NOT NULL	BOOLEAN	非空判断。 若value不为NULL则返回TRUE，否则返回FALSE；是IS NULL的逻辑取反。	SELECT 1 IS NOT NULL; -- TRUE
value1 BETWEEN value2 AND value3	BOOLEAN	范围判断（闭区间）。 若value1在[value2, value3]闭区间内则返回TRUE，即等价于value1 >= value2 AND value1 <= value3，包含两端边界值；value2应不大于value3，否则结果恒为FALSE；若任一操作数为NULL则返回NULL。	SELECT 2 BETWEEN 1 AND 3; -- TRUE
value1 NOT BETWEEN value2 AND value3	BOOLEAN	不在范围内判断。 若value1不在[value2, value3]闭区间内则返回TRUE，等价于value1 < value2 OR value1 > value3；是BETWEEN的逻辑取反；若任一操作数为NULL则返回NULL。	SELECT 5 NOT BETWEEN 1 AND 3; -- TRUE
value1 LIKE value2	BOOLEAN	模式匹配。 若value1匹配模式value2则返回TRUE；value1为被匹配字符串，value2为模式字符串；%匹配任意字符序列（包括空序列），*_匹配恰好一个字符；如需匹配%或* 本身，需使用 ESCAPE子句定义转义字符；若任一操作数为NULL则返回NULL。	SELECT 'hello' LIKE 'h%'; -- TRUE
value1 NOT LIKE value2	BOOLEAN	不匹配模式。 若value1不匹配模式value2则返回TRUE；通配符规则与LIKE相同；是LIKE的逻辑取反；若任一操作数为NULL则返回 NULL。	SELECT 'hello' NOT LIKE 'a%'; -- TRUE

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
value1 IN (value2[, value3]*)	BOOLEAN	在值列表中判断。 若value1等于列表中任意一个值则返回TRUE；value1与列表中值类型必须兼容；若value1不在列表中且列表包含NULL，则返回NULL而非FALSE；若value1为NULL则返回NULL。	SELECT 2 IN (1, 2, 3); -- TRUE
value1 NOT IN (value2[, value3]*)	BOOLEAN	不在值列表中判断。 若value1不等于列表中任何一个值则返回TRUE；是IN的逻辑取反；若列表包含NULL且value1不在列表的非NULL值中，则返回NULL而非TRUE；若value1为NULL则返回NULL。	SELECT 5 NOT IN (1, 2, 3); -- TRUE
value1 IS DISTINCT FROM value2	BOOLEAN	不同值判断（NULL安全）。 若value1和value2的值不同则返回TRUE；与<>的关键区别在于NULL处理：NULL IS DISTINCT FROM NULL返回FALSE，NULL IS DISTINCT FROM非NULL返回TRUE，不会产生NULL结果；适用于需要将NULL视为可比较值的场景。	SELECT 1 IS DISTINCT FROM 2; -- TRUE
value1 IS NOT DISTINCT FROM value2	BOOLEAN	相同值判断（NULL安全）。 若value1和value2的值相同则返回TRUE；与=的关键区别在于NULL处理：NULL IS NOT DISTINCT FROM NULL返回TRUE，不会产生NULL结果；是IS DISTINCT FROM的逻辑取反；适用于需要将NULL视为相等值的场景。	SELECT NULL IS NOT DISTINCT FROM NULL; -- TRUE

6.2 逻辑函数

概述

逻辑函数对布尔表达式进行逻辑运算，返回BOOLEAN类型结果。AI DataLake支持SQL标准中的AND、OR、NOT三值逻辑运算、IS TRUE/IS FALSE系列布尔判断以及IF条件函数，是构建复杂查询条件的基础。

版权所有 © 华为云计算技术有限公司

更多内容请参考**Logical Functions**。

适用场景

- **组合过滤条件**：在WHERE子句中使用AND/OR组合多个条件。
- **条件取反**：使用NOT对条件取反。
- **布尔值判断**：使用IS TRUE/IS FALSE安全判断布尔值，避免NULL干扰。
- **条件赋值**：使用IF函数根据条件返回不同的表达式值。

约束与限制

- AND和OR遵循SQL三值逻辑：当任一操作数为NULL时，结果可能为NULL。
- NOT NULL的结果仍为NULL，不会变为TRUE或FALSE。
- IS TRUE/IS NOT TRUE/IS FALSE/IS NOT FALSE对NULL始终返回确定的TRUE或FALSE，不产生NULL结果。
- IF函数的true_value和false_value参数类型必须兼容，否则需显式CAST转换。

语法结构

语法	返回类型	说明	示例
boolean1 OR boolean2	BOOLEAN	逻辑或。 任一操作数为TRUE则返回TRUE；两个操作数均为FALSE则返回FALSE；当含NULL时：若另一操作数为TRUE则返回TRUE，若另一操作数为FALSE则返回NULL。即TRUE OR NULL为TRUE，FALSE OR NULL为NULL。	SELECT TRUE OR FALSE; -- TRUE
boolean1 AND boolean2	BOOLEAN	逻辑与。 两个操作数均为TRUE则返回TRUE；任一操作数为FALSE则返回FALSE；当含NULL时：若另一操作数为FALSE则返回FALSE，若另一操作数为TRUE则返回NULL。即FALSE AND NULL为FALSE，TRUE AND NULL为NULL。	SELECT TRUE AND FALSE; -- FALSE
NOT boolean	BOOLEAN	逻辑非。 将TRUE变为FALSE，FALSE变为TRUE；NOT NULL的结果仍为NULL。	SELECT NOT TRUE; -- FALSE

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
boolean IS TRUE	BOOLEAN	判断是否为TRUE。 若boolean为TRUE则返回TRUE，若为FALSE则返回FALSE，若为NULL则返回FALSE。与直接使用boolean不同，IS TRUE对NULL返回确定的FALSE而非NULL。	SELECT NULL IS TRUE; -- FALSE
boolean IS NOT TRUE	BOOLEAN	判断是否不为TRUE。 若boolean为TRUE则返回FALSE，若为FALSE或NULL则返回TRUE。等价于NOT(boolean IS TRUE)，但结果始终为确定的TRUE或FALSE。	SELECT NULL IS NOT TRUE; -- TRUE
boolean IS FALSE	BOOLEAN	判断是否为FALSE。 若boolean为FALSE则返回TRUE，若为TRUE则返回FALSE，若为NULL则返回FALSE。对NULL返回确定的FALSE而非NULL。	SELECT NULL IS FALSE; -- FALSE
boolean IS NOT FALSE	BOOLEAN	判断是否不为FALSE。 若boolean为FALSE则返回FALSE，若为TRUE或NULL则返回TRUE。等价于NOT(boolean IS FALSE)，但结果始终为确定的TRUE或FALSE。	SELECT NULL IS NOT FALSE; -- TRUE
IF(condition, true_value, false_value)	与true_value/false_value一致	条件函数。 condition为BOOLEAN类型，若condition为TRUE则返回true_value，若condition为FALSE或NULL则返回false_value；true_value和false_value类型必须兼容。	SELECT IF(3 > 2, 'yes', 'no'); -- 'yes'

6.3 算术函数

概述

算术函数对数值类型进行数学运算，涵盖一元正负号、四则运算、取余、幂运算、开方、对数、指数、三角函数、取整截断、符号判断、数学常量、随机数及进制转换等完整的数学计算能力。

 版权所有 © 华为云计算技术有限公司

更多内容请参考**Arithmetic Functions**。

适用场景

- **数值计算**：在SELECT中对字段进行加减乘除运算，实现业务指标计算。
- **精度控制**：通过CEIL、FLOOR、ROUND、TRUNCATE进行数值精度转换和截断。
- **科学计算**：通过三角函数、对数、指数函数进行科学和工程计算。
- **数据转换**：通过BIN、HEX进行数值与进制字符串之间的转换。
- **随机采样**：通过RAND生成随机数用于数据采样或测试数据构造。

约束与限制

- 整数除法会截断小数部分，如需保留小数请使用DECIMAL类型或除以DECIMAL值。
- 除以零会产生NULL结果。
- POWER和SQRT的输入和返回类型为DOUBLE。
- LN、LOG10、LOG2的输入必须大于 0，否则返回NULL。
- 三角函数SIN、COS、TAN、COT的参数为弧度而非角度；反三角函数ASIN、ACOS的输入范围分别为 [-1, 1]，超出范围返回NULL。
- 若算术函数的任一输入为NULL，则结果为NULL。
- TRUNCATE直接截断而非四舍五入，与ROUND的舍入行为不同。

语法结构

语法	返回类型	说明	示例
+ numeric	与输入相同	一元正号。返回numeric本身的值，不改变数值；numeric为任意数值类型（TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、DECIMAL）。	`SELECT +5; -- 5`
- numeric	与输入相同	一元负号。返回numeric的相反数；numeric为任意数值类型。	`SELECT -5; -- -5`
numeric1 + numeric2	自动提升类型	加法运算。返回numeric1与numeric2的和；两操作数为任意数值类型，结果类型按SQL标准自动提升（如 INT + DOUBLE → DOUBLE）。	`SELECT 10 + 5; -- 15`
numeric1 - numeric2	自动提升类型	减法运算。返回numeric1减去numeric2的差；两操作数为任意数值类型，结果类型自动提升。	`SELECT 10 - 5; -- 5`

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
numeric1 * numeric2	自动提升类型	乘法运算。返回numeric1与numeric2的积；两操作数为任意数值类型，结果类型自动提升。	SELECT 10 * 5; -- 50
numeric1 / numeric2	自动提升类型	除法运算。返回numeric1除以numeric2的商；整数除法截断小数部分，如需保留小数请使用DECIMAL类型；除以零返回NULL。	SELECT 10 / 3; -- 3
POWER(numeric1, numeric2)	DOUBLE	幂运算。返回numeric1的numeric2次幂，即numeric1^numeric2；两参数均为数值类型，返回DOUBLE；若numeric1为负数且numeric2为非整数，结果为NULL。	SELECT POWER(2, 3); -- 8.0
ABS(numeric)	与输入相同	绝对值。返回numeric的绝对值；numeric为任意数值类型，返回类型与输入相同。	SELECT ABS(-5); -- 5
MOD(numeric1, numeric2)	与输入相同	取余运算。返回numeric1除以numeric2的余数，与%运算符等价；numeric2为0时返回NULL；两参数为整数类型时返回整数，否则返回DECIMAL。	SELECT MOD(10, 3); -- 1
SQRT(numeric)	DOUBLE	平方根。返回numeric的非负平方根；numeric必须为非负数值，为负时返回NULL。	SELECT SQRT(16); -- 4.0
LN(numeric)	DOUBLE	自然对数。返回numeric的自然对数（以 e 为底）；numeric必须大于 0，否则返回NULL。	SELECT LN(2.718281828); -- 0.9999999995
LOG10(numeric)	DOUBLE	以10为底的对数。返回numeric以10为底的对数；numeric必须大于 0，否则返回NULL。	SELECT LOG10(100); -- 2.0
LOG2(numeric)	DOUBLE	以2为底的对数。返回numeric以2为底的对数；numeric必须大于0，否则返回NULL。	SELECT LOG2(8); -- 3.0
LOG(numeric1, numeric2)	DOUBLE	自定义底数对数。返回以numeric2为底numeric1的对数；numeric1必须大于0，numeric2必须大于0且不等于1，否则返回NULL。	SELECT LOG(8, 2); -- 3.0

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
EXP(numeric)	DOUBLE	指数运算。返回e（自然常数）的numeric次幂，即e^numeric；是LN的反函数。	SELECT EXP(1); -- 2.718281828459045
CEIL(numeric) / CEILING(numeric)	与输入相同	向上取整。返回不小于numeric的最小整数；numeric为任意数值类型。	SELECT CEIL(3.2); -- 4
FLOOR(numeric)	与输入相同	向下取整。返回不大于numeric的最大整数；numeric为任意数值类型。	SELECT FLOOR(3.8); -- 3
ROUND(numeric, integer)	与输入相同	四舍五入。将numeric四舍五入到integer位小数；integer为正时保留对应小数位，为负时对整数位进行舍入，为0时舍入到整数。	SELECT ROUND(3.456, 2); -- 3.46
TRUNCATE(numeric, integer)	与输入相同	截断。将numeric截断到integer位小数，直接丢弃多余小数位而不进行舍入；integer为正时保留对应小数位，为负时对整数位截断，为0时截断到整数。	SELECT TRUNCATE(3.456, 2); -- 3.45
SIGN(numeric)	INT	符号函数。返回numeric的符号：numeric大于0返回 1，等于0返回 0，小于0返回-1。	SELECT SIGN(-5); -- -1
SIN(numeric)	DOUBLE	正弦函数。返回numeric（弧度）的正弦值；numeric为数值类型，表示弧度。	SELECT SIN(0); -- 0.0
COS(numeric)	DOUBLE	余弦函数。返回numeric（弧度）的余弦值；numeric为数值类型，表示弧度。	SELECT COS(0); -- 1.0
TAN(numeric)	DOUBLE	正切函数。返回numeric（弧度）的正切值；numeric为数值类型，表示弧度。	SELECT TAN(0); -- 0.0
COT(numeric)	DOUBLE	余切函数。返回numeric（弧度）的余切值，即 1/TAN(numeric)；numeric为0或π的整数倍时返回NULL。	SELECT COT(1); -- 0.6420926159343308
ASIN(numeric)	DOUBLE	反正弦函数。返回numeric的反正弦值（弧度）；numeric范围必须在 [-1, 1]，超出范围返回NULL。	SELECT ASIN(1); -- 1.5707963267948966

 版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
ACOS(numeric)	DOUBLE	反余弦函数。返回numeric的反余弦值（弧度）；numeric范围必须在 [-1, 1]，超出范围返回NULL。	SELECT ACOS(1); -- 0.0
ATAN(numeric)	DOUBLE	反正切函数。返回numeric的反正切值（弧度）；numeric为任意数值类型，返回值范围为 (-π/2, π/2)。	SELECT ATAN(1); -- 0.7853981633974483
ATAN2(numeric1, numeric2)	DOUBLE	双参数反正切函数。返回点 (numeric2, numeric1) 与正x轴之间的角度（弧度）；与ATAN的区别在于通过两个参数确定象限，返回值范围为 (-π, π]。	SELECT ATAN2(1, 1); -- 0.7853981633974483
PI()	DOUBLE	圆周率。返回π的近似值3.141592653589793；无参数。	SELECT PI(); -- 3.141592653589793
E()	DOUBLE	自然常数。返回e的近似值2.718281828459045；无参数。	SELECT E(); -- 2.718281828459045
RAND()	DOUBLE	随机数。返回 [0.0, 1.0) 范围内的伪随机DOUBLE 值；无参数，每次调用返回不同值。	SELECT RAND(); -- 0.123...（随机值）
RAND(integer)	DOUBLE	种子随机数。返回 [0.0, 1.0) 范围内的伪随机DOUBLE值；integer为种子值，相同种子产生相同的随机序列。	SELECT RAND(1); -- 0.730...（固定种子）
BIN(integer)	STRING	整数转二进制字符串。返回integer的二进制表示字符串；integer为整数类型（TINYINT、SMALLINT、INT、BIGINT），负数返回其补码的二进制表示。	SELECT BIN(5); -- '101'
HEX(numeric)	STRING	转十六进制字符串。返回numeric的十六进制表示字符串；numeric为整数类型时返回其十六进制表示，为STRING类型时将每个字符转为两位十六进制。	SELECT HEX(255); -- 'FF'
BITAND	INT	BITAND函数用于执行按位与操作。按位与操作是对两个整数的二进制位逐位进行比较，当两个对应位均为1时结果位为1，否则为0。	详情请参见**8.1.1 BITAND**

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
BITNOT	INT	BITNOT函数用于执行按位取反操作。按位取反操作是将整数的二进制位逐位取反，即0变1、1变0。由于INT类型采用补码表示，取反后的结果需结合补码规则理解。	详情请参见**8.1.2 BITNOT**
BITOR	INT	BITOR函数用于执行按位或操作。按位或操作是对两个整数的二进制位逐位进行比较，当两个对应位中至少有一个为1时结果位为1，否则为0。	详情请参见**8.1.3 BITOR**
BITXOR	INT	BITXOR函数用于执行按位异或操作。按位异或操作是对两个整数的二进制位逐位进行比较，当两个对应位不同时结果位为1，相同时为0。	详情请参见**8.1.4 BITXOR**

6.4 字符串函数

概述

字符串函数用于对VARCHAR/STRING类型数据进行操作，包括拼接、截取、替换、大小写转换、修剪、长度计算、填充、模式匹配等。

更多内容请参考**String Functions**。

适用场景

- **数据清洗**：通过TRIM去除空格，通过UPPER/LOWER统一大小写。
- **字符串拼接**：通过CONCAT或CONCAT_WS拼接多个字段。
- **数据脱敏**：通过SUBSTRING和REPLACE实现字段脱敏处理。
- **模式替换**：通过REGEXP_REPLACE进行正则替换。
- **格式化输出**：通过LPAD/RPAD对齐字段，通过FORMAT_NUMBER格式化数字。
- **模式匹配**：通过LIKE进行简单模式匹配，通过REGEXP进行正则匹配。

约束与限制

- CONCAT中任一参数为NULL时结果为NULL，CONCAT_WS会跳过NULL参数。
- SUBSTRING的起始位置从1开始，0与1等价。
- REGEXP_REPLACE使用Java正则语法。
- TRIM指定string1时仅去除与string1完全匹配的单字符，非子串删除。
- 当len小于string原始长度时，LPAD/RPAD会截断字符串到len长度。

版权所有 © 华为云计算技术有限公司

语法结构

语法	返回类型	说明	示例
string1 || string2	STRING	字符串拼接。将string1和string2首尾连接为一个新字符串；string1和string2均须为STRING类型；若任一操作数为NULL则返回NULL。	SELECT 'hello' || ' ' || 'world'; -- 'hello world'
CHAR_LENGTH(string)	INT	返回字符串的字符数。string须为STRING类型；与CHARACTER_LENGTH完全等价；多字节字符（如中文）按1个字符计数；若string为NULL则返回NULL。	SELECT CHAR_LENGTH('hello'); -- 5
CHARACTER_LENGTH(string)	INT	返回字符串的字符数。与CHAR_LENGTH完全等价，为SQL标准别名；string须为STRING类型；若string为NULL则返回NULL。	SELECT CHARACTER_LENGTH('你好'); -- 2
UPPER(string)	STRING	转换为大写。将string中所有小写字母转换为大写字母，非字母字符保持不变；string须为STRING类型；若string为NULL则返回NULL。	SELECT UPPER('hello'); -- 'HELLO'
LOWER(string)	STRING	转换为小写。将string中所有大写字母转换为小写字母，非字母字符保持不变；string须为STRING类型；若string为NULL则返回NULL。	SELECT LOWER('HELLO'); -- 'hello'
INITCAP(string)	STRING	首字母大写。将string中每个单词的首字母转换为大写，其余字母转换为小写；单词以非字母字符分隔；string须为STRING类型；若string为NULL则返回NULL。	SELECT INITCAP('hello world'); -- 'Hello World'

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
TRIM([BOTH | LEADING | TRAILING] string1 FROM string2)	STRING	去除指定字符。从string2的首部（LEADING）、尾部（TRAILING）或两端（BOTH，默认）去除与string1匹配的字符；省略string1和BOTH时默认去除两端空格；string1和string2均须为STRING类型；若任一参数为NULL则返回NULL。	SELECT TRIM(BOTH 'x' FROM 'xxhelloxx'); -- 'hello'
LTRIM(string)	STRING	去除前导空格。移除string左侧所有空格字符；string须为STRING类型；若string为NULL则返回NULL。	SELECT LTRIM(' hello'); -- 'hello'
RTRIM(string)	STRING	去除尾部空格。移除string右侧所有空格字符；string须为STRING类型；若string为NULL则返回NULL。	SELECT RTRIM('hello '); -- 'hello'
SUBSTRING(string FROM pos [FOR len])	STRING	截取子串（SQL标准语法）。从string的pos位置开始截取长度为len的子串；pos从1开始，0与1等价；省略FOR len时截取到字符串末尾；pos为负数时从末尾倒数；string须为STRING类型，pos须为INT类型；若string为NULL则返回NULL。	SELECT SUBSTRING('hello world' FROM 1 FOR 5); -- 'hello'
SUBSTRING(string, pos [, len])	STRING	截取子串（逗号语法）。与SUBSTRING(string FROM pos [FOR len])语义完全等价，仅语法形式不同；pos从1开始，0与1等价；省略len时截取到字符串末尾；string须为STRING类型，pos须为INT类型；若string为NULL则返回NULL。	SELECT SUBSTRING('hello world', 7); -- 'world'
CONCAT(string1, string2, ...)	STRING	拼接字符串。将所有参数按顺序连接为一个字符串；所有参数均须为STRING类型；支持两个及以上参数；与||运算符的关键区别：任一参数为NULL时整个结果为NULL。	SELECT CONCAT('hello', ' ', 'world'); -- 'hello world'

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
CONCAT_WS(separator, string1, ...)	STRING	用分隔符拼接。使用separator作为分隔符连接所有非NULL的string参数；separator和string参数均须为STRING类型；与CONCAT的关键区别：跳过NULL参数而非返回NULL；若separator为NULL则返回NULL。	SELECT CONCAT_WS('-', '2024', '01', '15'); -- '2024-01-15'
REPLACE(string, old, new)	STRING	替换子串。将string中所有出现的old子串替换为new；区分大小写；old为空字符串时不做替换；string、old、new均须为STRING类型；若任一参数为NULL则返回NULL。	SELECT REPLACE('hello world', 'world', 'Flink'); -- 'hello Flink'
REGEXP_REPLACE(string, regex, replacement)	STRING	正则替换。使用Java正则语法将string中所有匹配regex的子串替换为replacement；replacement中可使用1、1、2等引用捕获组；string、regex、replacement均须为STRING类型；若任一参数为NULL则返回NULL。	SELECT REGEXP_REPLACE('abc123', '[0-9]', '*'); -- 'abc***'
REGEXP_EXTRACT(string, regex [, group])	STRING	正则提取。从string中提取匹配regex的子串；group指定返回的捕获组编号，默认为0（返回整个匹配）；string和regex须为STRING类型，group须为INT类型；若无匹配则返回NULL；若任一参数为NULL则返回NULL。	SELECT REGEXP_EXTRACT('abc123def', '([a-z]+)([0-9]+)', 2); -- '123'
POSITION(string1 IN string2)	INT	查找位置。返回string1在string2中首次出现的起始位置，从1开始计数；若未找到则返回0；string1和string2均须为STRING类型；若任一参数为NULL则返回NULL。	SELECT POSITION('world' IN 'hello world'); -- 7

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
LPAD(string, len, pad)	STRING	左侧填充。在string左侧填充pad字符串直到总长度达到len；若string长度已超过len则从右侧截断至len长度；string和pad须为STRING类型，len须为INT类型；若任一参数为NULL则返回NULL。	SELECT LPAD('5', 3, '0'); -- '005'
RPAD(string, len, pad)	STRING	右侧填充。在string右侧填充pad字符串直到总长度达到len；若string长度已超过len则从右侧截断至len长度；string和pad须为STRING类型，len须为INT类型；若任一参数为NULL则返回NULL。	SELECT RPAD('5', 3, '0'); -- '500'
OVERLAY(string1 PLACING string2 FROM pos [FOR len])	STRING	覆盖子串。从string1的pos位置开始，用string2覆盖len个字符；pos从1开始；省略FOR len时覆盖长度为string2的长度；string1和string2须为STRING类型，pos和len须为INT类型；若任一参数为NULL则返回NULL。	SELECT OVERLAY('hello world' PLACING 'FLINK' FROM 7); -- 'hello FLINK'
REVERSE(string)	STRING	反转字符串。将string中字符顺序完全反转；string须为STRING类型；若string为NULL则返回NULL。	SELECT REVERSE('hello'); -- 'olleh'
REPEAT(string, integer)	STRING	重复字符串。将string重复integer次后连接为一个新字符串；string须为STRING类型，integer须为INT类型；integer为0时返回空字符串，integer为负数时返回NULL；若string为NULL则返回NULL。	SELECT REPEAT('ab', 3); -- 'ababab'
SPACE(integer)	STRING	生成空格字符串。返回由integer个空格字符组成的字符串；integer须为INT类型；integer为0时返回空字符串，integer为负数时返回NULL。	SELECT SPACE(3); -- ' '

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
LEFT(string, integer)	STRING	左侧截取。返回string最左侧的integer个字符；string须为STRING类型，integer须为INT类型；integer为0时返回空字符串，integer为负数时返回NULL；若string为NULL则返回NULL。	SELECT LEFT('hello world', 5); -- 'hello'
RIGHT(string, integer)	STRING	右侧截取。返回string最右侧的integer个字符；string须为STRING类型，integer须为INT类型；integer为0时返回空字符串，integer为负数时返回NULL；若string为NULL则返回NULL。	SELECT RIGHT('hello world', 5); -- 'world'
FORMAT_NUMBER(numeric, decimal_places)	STRING	格式化数字。将numeric格式化为带千分位分隔符的字符串，保留decimal_places位小数；numeric须为数值类型，decimal_places须为INT类型；若任一参数为NULL则返回NULL。	SELECT FORMAT_NUMBER(12345.6789, 2); -- '12,345.68'
LIKE(string1, string2)	BOOLEAN	模式匹配。判断string1是否匹配模式string2；%匹配任意字符序列（包括空序列），_匹配恰好一个字符；如需匹配通配符本身需使用ESCAPE子句转义；string1和string2均须为STRING类型；若任一参数为NULL则返回NULL。	SELECT LIKE('hello', 'h%'); -- TRUE
REGEXP(string1, string2)	BOOLEAN	正则匹配。判断string1是否匹配正则表达式string2；使用Java正则语法；string1和string2均须为STRING类型；若任一参数为NULL则返回NULL。	SELECT REGEXP('abc123', '[a-z]+[0-9]+'); -- TRUE
URL_DECODE	VARCHAR	URL_DECODE函数用于将application/x-www-form-urlencoded MIME格式的输入字符串转换为标准字符串。	详情请参见**8.1.8 URL_DECODE**。

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
URL_ENCODE	VARCHAR	URL_ENCODE用于以application/x-www-form-urlencoded MIME格式对输入字符串进行编码并返回编码后的字符串。	详情请参见8.1.9 URL_ENCODE。

6.5 时间函数

概述

时间函数用于对DATE、TIME、TIMESTAMP类型数据进行操作，包括类型转换、获取当前时间、提取时间分量、格式化、时间取整、时间偏移计算等。

更多内容请参考Temporal Functions。

适用场景

- **Watermark定义**：通过CURRENT_TIMESTAMP和INTERVAL定义Watermark策略。
- **窗口分组**：通过DATE_FORMAT将时间格式化为小时/天级别用于窗口分组。
- **时间计算**：通过TIMESTAMPADD/TIMESTAMPDIFF计算时间偏移和差值。
- **ETL时间字段处理**：提取年月日等分量用于数据分区和聚合。

约束与限制

- CURRENT_TIMESTAMP返回的是UTC时区的时间戳，LOCALTIMESTAMP返回的是本地时区的时间戳。
- DATE_FORMAT使用Java DateTimeFormatter格式模式（如yyyy-MM-dd、HH:mm:ss），与SimpleDateFormat模式有部分差异。
- INTERVAL支持年月间隔（YEAR、MONTH）和日时间隔（DAY、HOUR、MINUTE、SECOND）两种类型，不能混用。
- PROCTIME()只能用于DDL中声明处理时间属性，不能用于查询中直接调用。

类型转换类语法结构

语法	返回类型	说明	示例
DATE string	DATE	将字符串转换为DATE类型。参数string必须为DATE或STRING类型，格式须为yyyy-MM-dd，否则返回NULL。返回值为DATE类型。	SELECT DATE '2024-01-15'; -- 2024-01-15

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
TIME string	TIME	将字符串转换为TIME类型。参数string必须为TIME或STRING类型，格式须为HH:mm:ss，否则返回NULL。返回值为TIME类型。	SELECT TIME '10:30:00'; -- 10:30:00
TIMESTAMP string	TIMESTAMP	将字符串转换为TIMESTAMP类型。参数string必须为TIMESTAMP或STRING类型，格式须为yyyy-MM-dd HH:mm:ss[.SSSSSS]，否则返回NULL。返回值为TIMESTAMP类型。	SELECT TIMESTAMP '2024-01-15 10:30:00'; -- 2024-01-15 10:30:00
INTERVAL string range	INTERVAL	将字符串转换为时间间隔。参数string为表示间隔的STRING类型，range为单位限定符，可选YEAR、MONTH、DAY、HOUR、MINUTE、SECOND、YEAR TO MONTH、DAY TO SECOND等。返回值为INTERVAL类型。	SELECT INTERVAL '1' DAY; -- +1 00:00:00.000
TO_DATE(string[, format])	DATE	将字符串按指定格式转换为DATE类型。参数string为STRING类型，format为可选的Java DateTimeFormatter格式字符串，默认yyyy-MM-dd。格式不匹配时返回NULL。返回值为DATE类型。	SELECT TO_DATE('2024-01-15', 'yyyy-MM-dd'); -- 2024-01-15
TO_TIMESTAMP(string[, format])	TIMESTAMP	将字符串按指定格式转换为TIMESTAMP类型。参数string为STRING类型，format为可选的Java DateTimeFormatter格式字符串，默认yyyy-MM-dd HH:mm:ss。格式不匹配时返回NULL。返回值为TIMESTAMP类型。	SELECT TO_TIMESTAMP('2024-01-15 10:30:00', 'yyyy-MM-dd HH:mm:ss'); -- 2024-01-15 10:30:00

版权所有 © 华为云计算技术有限公司

获取当前时间类语法结构

语法	返回类型	说明	示例
CURRENT_DATE	DATE	返回当前SQL会话的日期（UTC时区）。无参数。返回值为DATE类型。	SELECT CURRENT_DATE; -- 2024-01-15
CURRENT_TIME	TIME WITH TIME ZONE	返回当前SQL会话的时间（带时区）。无参数。返回值为TIME WITH TIME ZONE类型。	SELECT CURRENT_TIME; -- 10:30:00+08:00
CURRENT_TIMESTAMP	TIMESTAMP_LTZ(3)	返回当前SQL会话的时间戳（带本地时区，精度为3位毫秒）。无参数。返回值为TIMESTAMP_LTZ(3)类型。	SELECT CURRENT_TIMESTAMP; -- 2024-01-15T10:30:00.000+08:00
LOCALTIME	TIME	返回当前SQL会话的本地时间（不带时区）。无参数。返回值为TIME类型。	SELECT LOCALTIME; -- 10:30:00
LOCALTIMESTAMP	TIMESTAMP(3)	返回当前SQL会话的本地时间戳（不带时区，精度为3位毫秒）。无参数。返回值为TIMESTAMP(3)类型。	SELECT LOCALTIMESTAMP; -- 2024-01-15 10:30:00.000
NOW()	TIMESTAMP_LTZ(3)	返回当前时间戳，与CURRENT_TIMESTAMP等价。无参数。返回值为TIMESTAMP_LTZ(3)类型。	SELECT NOW(); -- 2024-01-15T10:30:00.000+08:00

时间分量提取类语法结构

语法	返回类型	说明	示例
EXTRACT(timeintervalunit FROM temporal)	INT	从时间值中提取指定时间部分的值。参数timeintervalunit为时间单位枚举，可选YEAR、MONTH、DAY、HOUR、MINUTE、SECOND、QUARTER、WEEK、DOW、DOY等；参数temporal必须为DATE、TIME、TIMESTAMP或INTERVAL类型。返回值为INT类型。	SELECT EXTRACT(YEAR FROM TIMESTAMP '2024-01-15 10:30:00'); -- 2024
YEAR(date)	INT	提取日期中的年份。参数date必须为DATE或TIMESTAMP类型。返回值为INT类型，范围1~9999。	SELECT YEAR(DATE '2024-01-15'); -- 2024

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
QUARTER(date)	INT	提取日期中的季度。参数date必须为DATE或TIMESTAMP类型。返回值为INT类型，范围1~4。	SELECT QUARTER(DATE '2024-04-15'); -- 2
MONTH(date)	INT	提取日期中的月份。参数date必须为DATE或TIMESTAMP类型。返回值为INT类型，范围1~12。	SELECT MONTH(DATE '2024-01-15'); -- 1
WEEK(date)	INT	提取日期在年中的周数。参数date必须为DATE或TIMESTAMP类型。返回值为INT类型，范围1~53。	SELECT WEEK(DATE '2024-01-15'); -- 3
DAYOFYEAR(date)	INT	提取日期在年中的天数。参数date必须为DATE或TIMESTAMP类型。返回值为INT类型，范围1~366。	SELECT DAYOFYEAR(DATE '2024-02-01'); -- 32
DAYOFMONTH(date)	INT	提取日期在月中的天数。参数date必须为DATE或TIMESTAMP类型。返回值为INT类型，范围1~31。	SELECT DAYOFMONTH(DATE '2024-01-15'); -- 15
DAYOFWEEK(date)	INT	提取日期在周中的天数。参数date必须为DATE或TIMESTAMP类型。返回值为INT类型，范围1（周日）~7（周六）。	SELECT DAYOFWEEK(DATE '2024-01-15'); -- 2
HOUR(timestamp)	INT	提取时间戳中的小时部分。参数timestamp必须为TIME或TIMESTAMP类型。返回值为INT类型，范围0~23。	SELECT HOUR(TIMESTAMP '2024-01-15 10:30:00'); -- 10
MINUTE(timestamp)	INT	提取时间戳中的分钟部分。参数timestamp必须为TIME或TIMESTAMP类型。返回值为INT类型，范围0~59。	SELECT MINUTE(TIMESTAMP '2024-01-15 10:30:00'); -- 30
SECOND(timestamp)	INT	提取时间戳中的秒部分。参数timestamp必须为TIME或TIMESTAMP类型。返回值为INT类型，范围0~59。	SELECT SECOND(TIMESTAMP '2024-01-15 10:30:45'); -- 45

版权所有 © 华为云计算技术有限公司

时间取整类语法结构

语法	返回类型	说明	示例
FLOOR(temporal TO timeintervalunit)	与输入类型一致	将时间值向下取整到指定时间单位。参数temporal必须为DATE、TIME或TIMESTAMP类型；参数timeintervalunit为取整单位，可选YEAR、MONTH、DAY、HOUR、MINUTE、SECOND等。返回值类型与输入一致。	SELECT FLOOR(TIMESTAMP '2024-01-15 10:30:45' TO HOUR); -- 2024-01-15 10:00:00
CEIL(temporal TO timeintervalunit)	与输入类型一致	将时间值向上取整到指定时间单位。参数temporal必须为DATE、TIME或TIMESTAMP类型；参数timeintervalunit为取整单位，可选YEAR、MONTH、DAY、HOUR、MINUTE、SECOND等。返回值类型与输入一致。	SELECT CEIL(TIMESTAMP '2024-01-15 10:30:45' TO HOUR); -- 2024-01-15 11:00:00

时间格式化类

语法	返回类型	说明	示例
DATE_FORMAT(timestamp, format)	STRING	将时间戳按指定格式转为字符串。参数timestamp必须为TIMESTAMP或TIMESTAMP_LTZ类型；参数format为STRING类型，使用Java DateTimeFormatter格式模式（如yyyy-MM-dd、HH:mm:ss、yyyyMMdd等）。返回值为STRING类型。	SELECT DATE_FORMAT(TIMESTAMP '2024-01-15 10:30:00', 'yyyy-MM-dd'); -- '2024-01-15'

版权所有 © 华为云计算技术有限公司

时间运算类

语法	返回类型	说明	示例
TIMESTAMPADD(unit, interval, timestamp)	TIMESTAMP	对时间戳增加指定间隔。参数unit为时间单位，可选YEAR、MONTH、QUARTER、WEEK、DAY、HOUR、MINUTE、SECOND等；参数interval必须为INT类型，表示要增加的数量（可为负数实现减法）；参数timestamp必须为TIMESTAMP类型。返回值为TIMESTAMP类型。	SELECT TIMESTAMPADD(DAY, 7, TIMESTAMP '2024-01-15 10:30:00'); -- 2024-01-22 10:30:00
TIMESTAMPDIFF(unit, timestamp1, timestamp2)	INT	计算两个时间戳之间的差值。参数unit为时间单位，可选YEAR、MONTH、QUARTER、WEEK、DAY、HOUR、MINUTE、SECOND等；参数timestamp1和timestamp2必须为TIMESTAMP类型。返回值为INT类型，表示timestamp2 - timestamp1在指定单位下的差值。	SELECT TIMESTAMPDIFF(DAY, TIMESTAMP '2024-01-01 00:00:00', TIMESTAMP '2024-01-15 00:00:00'); -- 14
DATE_ADD	STRING	DATE_ADD函数用于返回指定日期增加目标天数后的日期。当目标天数为负数时，相当于减去对应天数。	详情请参见**8.1.5 DATE_ADD**。
DATE_SUB	STRING	DATE_SUB函数用于返回指定日期减去目标天数后的日期。当目标天数为负数时，相当于增加对应天数。	详情请参见**8.1.6 DATE_SUB**。
DATEDIFF	INT	DATEDIFF函数用于计算两个日期之间的天数差值。计算方式为enddate减去startdate，即返回值 = enddate - startdate。当enddate晚于startdate时返回正数，当enddate早于startdate时返回负数。	详情请参见**8.1.7 DATEDIFF**。

版权所有 © 华为云计算技术有限公司

时间属性处理类

语法	返回类型	说明	示例
PROCTIME()	TIMESTAMP_LTZ(3)	声明处理时间属性，返回当前处理时间戳。无参数。仅可在DDL的表定义中作为列的类型声明使用，不能在查询中直接调用。返回值为TIMESTAMP_LTZ(3)类型，表示数据被处理时的机器时间。	-- 在DDL中使用: proctime AS PROCTIME()

6.6 条件函数

概述

条件函数根据条件表达式返回不同的值。CASE WHEN支持多条件分支逻辑，IF实现简单的条件取值，COALESCE用于空值替换，NULLIF用于等值置空，ISNULL/ISNOTNULL用于空值判断，GREATEST/LEAST用于多值极值选取。AI DataLake支持完整的条件函数能力。

更多内容请参考**Conditional Functions**。

适用场景

- **数据分类**：通过CASE WHEN将连续值映射为离散分类，如将金额划分为高/中/低档位。
- **条件取值**：通过IF函数快速在两个值之间选择，如根据标志位返回不同状态文本。
- **空值替换**：通过COALESCE将NULL替换为默认值，避免NULL参与运算导致结果为NULL。
- **等值置空**：通过NULLIF在两个值相等时返回NULL，典型用法是避免除零错误。
- **空值判断**：通过ISNULL/ISNOTNULL在WHERE条件或CASE WHEN中判断值是否为NULL。
- **极值选取**：通过GREATEST/LEAST在多个列或表达式中选取最大或最小值，自动跳过NULL。

约束与限制

- 所有WHEN/THEN和ELSE的返回值类型必须兼容，系统会自动推导公共类型；若缺少ELSE子句且无条件满足，则返回NULL。
- true_value与false_value的类型必须兼容，系统会自动推导公共类型；condition必须为BOOLEAN类型。
- 要求至少两个参数；所有参数类型必须兼容，返回类型为所有参数的公共类型。
- value1和value2类型必须可比较；仅在value1 = value2为TRUE时返回NULL，若比较结果为NULL则返回value1。
- 要求至少两个参数；所有参数类型必须兼容且可排序；NULL值会被自动跳过，若所有值均为NULL则返回NULL。

版权所有 © 华为云计算技术有限公司

语法结构

语法	返回类型	说明	示例
CASE WHEN condition1 THEN result1 [WHEN condition2 THEN result2]* [ELSE result] END	与result一致	搜索型条件分支。按条件顺序依次求值，返回第一个满足条件（condition 为 TRUE）的result；若所有条件均不满足且存在 ELSE子句则返回ELSE的值；若所有条件均不满足且无ELSE子句则返回NULL；condition必须为BOOLEAN类型；所有 result类型必须兼容。	SELECT CASE WHEN 1 > 2 THEN 'a' WHEN 2 > 1 THEN 'b' ELSE 'c' END; -- 'b'
CASE value WHEN value1 THEN result1 [WHEN value2 THEN result2]* [ELSE result] END	与result一致	值匹配型条件分支。将value依次与每个WHEN后的valueN做等值比较，返回第一个匹配的result；若均不匹配且存在 ELSE 子句则返回ELSE的值；若均不匹配且无ELSE子句则返回 NULL；value与各valueN类型必须可比较；所有result类型必须兼容。	SELECT CASE 2 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'other' END; -- 'two'
IF(condition, true_value, false_value)	与true_value/false_value的公共类型一致	条件取值函数。若condition为TRUE则返回true_value，若condition为FALSE则返回false_value；condition必须为BOOLEAN类型；true_value与false_value类型必须兼容；若condition为NULL则返回false_value。	SELECT IF(1 > 2, 'a', 'b'); -- 'b'
COALESCE(value1, value2, ...)	与第一个非NULL值一致	返回参数列表中第一个非NULL的值；要求至少两个参数；所有参数类型必须兼容；若所有参数均为NULL则返回NULL。	SELECT COALESCE(NULL, NULL, 'default'); -- 'default'
NULLIF(value1, value2)	与value1一致或NULL	等值置空函数。若value1等于value2则返回NULL，否则返回value1；value1和value2 类型必须可比较；常用于避免除零错误（如 amount / NULLIF(count, 0)）。	SELECT NULLIF(5, 5); -- NULL
ISNULL(value)	BOOLEAN	空值判断函数。若value为NULL则返回TRUE，否则返回FALSE；等价于value IS NULL；不受NULL参与常规比较返回NULL的规则影响。	SELECT ISNULL(NULL); -- TRUE

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
ISNOTNULL(value)	BOOLEAN	非空判断函数。若value不为NULL 则返回TRUE，否则返回FALSE；等价于value IS NOT NULL；是ISNULL的逻辑取反。	SELECT ISNOTNULL(1); -- TRUE
GREATEST(value1, value2, ...)	与参数公共类型一致	返回参数列表中的最大值，自动跳过NULL；要求至少两个参数；所有参数类型必须兼容且可排序；若所有值均为NULL则返回 NULL。	SELECT GREATEST(3, 1, 4, 1, 5); -- 5
LEAST(value1, value2, ...)	与参数公共类型一致	返回参数列表中的最小值，自动跳过NULL；要求至少两个参数；所有参数类型必须兼容且可排序；若所有值均为NULL则返回NULL。	SELECT LEAST(3, 1, 4, 1, 5); -- 1

6.7 类型转换函数

概述

类型转换函数用于在Flink SQL中实现不同数据类型之间的转换。通过这些函数，可以将一个数据类型的值转换为另一个数据类型，或者在运行时获取表达式的数据类型信息。类型转换是数据加工、清洗和格式适配中的核心操作，广泛用于ETL流程、异构数据源对接及动态类型判断等场景。

更多内容参考**Type Conversion Functions**。

适用场景

- **ETL数据清洗**：从异构数据源读取数据后，使用CAST或TRY_CAST将字符串字段转换为数值、日期等目标类型。
- **数据质量容错**：对可能存在脏数据的字段使用TRY_CAST，避免因个别转换失败导致整条记录处理中断。
- **动态类型判断**：在运行时通过TYPEOF获取字段或表达式的数据类型，用于条件分支或调试排障。
- **跨源数据对齐**：不同数据源的字段类型不一致时，通过类型转换函数统一数据类型后再进行JOIN或聚合。

约束与限制

- CAST和TRY_CAST的目标类型必须为Flink SQL支持的数据类型关键字，如INT、BIGINT、STRING、DOUBLE、DECIMAL、TIMESTAMP等。
- 并非所有类型之间均可互相转换，例如ARRAY与MAP之间、ROW与INT之间不支持直接转换，转换不支持的类型对会抛出异常（CAST）或返回NULL（TRY_CAST）。

版权所有 © 华为云计算技术有限公司

- 从STRING转换为数值类型时，字符串必须符合目标类型的格式要求，如CAST('abc' AS INT) 会抛出异常，TRY_CAST('abc' AS INT) 返回NULL。
- 从数值类型转换为STRING时，结果格式由系统默认决定，可能不保留原始精度信息。
- TYPEOF返回的类型字符串为Flink内部类型表示，其格式可能随版本变化，不应在业务逻辑中硬编码依赖该字符串的具体内容。
- TYPEOF(input, forceSerializable) 中forceSerializable为BOOLEAN类型，当设为TRUE时返回的类型字符串保证可序列化，但可能省略部分不可序列化的类型细节。

类型转换语法结构

语法	返回类型	说明	示例
CAST(value AS type)	由type决定	将value转换为type指定的数据类型。value为待转换的表达式，type为目标数据类型关键字（如INT、BIGINT、STRING、DOUBLE、DECIMAL(p, s)、BOOLEAN、DATE、TIMESTAMP 等）。转换失败时抛出运行时异常。支持所有Flink SQL数据类型之间的合法转换，不支持的类型对转换会直接报错。	SELECT CAST('123' AS INT); -- 123
TRY_CAST(value AS type)	由type决定，转换失败时返回 NULL	将value转换为type指定的数据类型。参数含义与CAST相同。与CAST的区别在于：当转换失败时（如类型不兼容或值格式不匹配），不抛出异常，而是返回NULL。适用于数据质量不确定、需要容错处理的场景。	SELECT TRY_CAST('abc' AS INT); -- NULL

类型查询语法结构

语法	返回类型	说明	示例
TYPEOF(input)	STRING	返回input表达式的数据类型字符串表示。input为任意类型的表达式。返回值反映了Flink内部对该表达式类型的信息，格式如INTEGER、VARCHAR(2147483647)、ROW<f1 INT, f2 STRING> 等。	SELECT TYPEOF(123); -- INTEGER

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
TYPEOF(input, forceSerializable)	STRING	返回input表达式的数据类型字符串表示。input为任意类型的表达式，forceSerializable为BOOLEAN类型。当forceSerializable为TRUE时，返回的类型字符串保证可序列化，适用于需要将类型信息在网络中传输或持久化的场景；为 FALSE时与单参数版本行为一致。	SELECT TYPEOF(123, TRUE); -- INTEGER

6.8 聚合函数

概述

聚合函数对一组输入值进行计算，返回单个结果值。聚合函数通常与GROUP BY子句配合使用，将数据按分组键聚合后计算每组的结果；也可作为窗口聚合函数在 GROUP WINDOW 或OVER WINDOW中使用。所有聚合函数在计算时默认跳过NULL值（ COUNT(*) 除外 ），支持通过DISTINCT关键字去除重复值后再聚合。

更多内容请参考**Aggregate Functions**。

适用场景

- 按维度分组统计指标，如按地区统计订单数量、平均金额。
- 窗口聚合计算滚动/滑动指标，如最近5分钟的UV、GMV。
- 数据去重统计，如COUNT(DISTINCT user_id) 计算独立访客数。
- 多值字段聚合，如将分组内标签收集为集合、将多行文本拼接为一条。
- ROLLUP/CUBE 多维分析中标识聚合层级。

约束与限制

- **NULL值处理**：除 COUNT(*) 外，所有聚合函数在计算时自动跳过NULL值；若分组内所有值均为NULL，聚合结果为NULL。
- **DISTINCT限制**：DISTINCT仅支持COUNT、AVG、SUM、COLLECT四个函数，MAX、MIN、STDDEV_POP、STDDEV_SAMP、VAR_POP、VAR_SAMP、LISTAGG、GROUPING、BIT_AND、BIT_OR不支持DISTINCT。
- **嵌套聚合**：聚合函数不可嵌套使用，如SUM(COUNT(*)) 不合法，需通过子查询实现。
- **GROUP BY一致性**：在SELECT中引用非聚合列时，该列必须出现在GROUP BY子句中。
- **窗口聚合**：聚合函数在OVER WINDOW中使用时，须配合ORDER BY和窗口范围（ ROWS BETWEEN / RANGE BETWEEN ）定义。
- **STDDEV_SAMP / VAR_SAMP最低行数**：样本标准差和样本方差要求分组内至少有2个非NULL值，否则返回NULL。

版权所有 © 华为云计算技术有限公司

- **BIT_AND / BIT_OR类型约束**：仅支持整数类型输入，传入非整数类型将导致类型校验失败。
- **COLLECT返回类型**：COLLECT返回的是MULTISET类型，如需转为数组可使用ARRAY_DISTINCT(COLLECT(...)) 等方式。
- **LISTAGG分隔符**：separator参数须为字符串常量，不支持列引用或表达式。

语法结构

语法	返回类型	说明	示例
COUNT(*)	BIGINT	统计所有行数，**包含NULL值的行**。无需指定列，对整行计数。	SELECT COUNT(*) FROM orders; -- 10
COUNT(value)	BIGINT	统计value非NULL的行数。参数可为任意类型；value为NULL的行不计入。	SELECT COUNT(amount) FROM orders; -- 8（跳过2条NULL）
COUNT(DISTINCT value)	BIGINT	统计value去NULL并去重后的不同值个数。参数可为任意类型。	SELECT COUNT(DISTINCT user_id) FROM orders; -- 5
AVG(value)	DECIMAL / DOUBLE	计算value的算术平均值。参数须为数值类型（TINYINT / SMALLINT / INT / BIGINT / FLOAT / DOUBLE / DECIMAL）；整型输入返回 DECIMAL，浮点输入返回DOUBLE；跳过 NULL。	SELECT AVG(price) FROM orders; -- 25.50
AVG(DISTINCT value)	DECIMAL / DOUBLE	计算value去NULL并去重后的算术平均值。参数须为数值类型；返回类型规则同AVG。	SELECT AVG(DISTINCT price) FROM orders; -- 30.00
SUM(value)	与输入类型一致	计算value的总和。参数须为数值类型；整型输入返回整型（INT→INT, BIGINT→BIGINT），DECIMAL→DECIMAL，FLOAT/DOUBLE→DOUBLE；跳过NULL。	SELECT SUM(amount) FROM orders; -- 1500
SUM(DISTINCT value)	与输入类型一致	计算value去NULL并去重后的总和。参数须为数值类型；返回类型规则同SUM。	SELECT SUM(DISTINCT amount) FROM orders; -- 1200

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
MAX(value)	与输入类型一致	返回value的最大值。参数可为任意可比较类型（数值、字符串、日期时间等）；跳过NULL；字符串按字典序比较。	SELECT MAX(price) FROM orders; -- 99.90
MIN(value)	与输入类型一致	返回value的最小值。参数可为任意可比较类型（数值、字符串、日期时间等）；跳过NULL；字符串按字典序比较。	SELECT MIN(price) FROM orders; -- 5.00
STDDEV_POP(value)	DOUBLE	计算value的**总体标准差**（Population Standard Deviation）。参数须为数值类型；公式为 $\sqrt{(\Sigma(x-\mu)^2/N)}$；跳过NULL；当输入行数 < 1 时返回NULL。	SELECT STDDEV_POP(score) FROM exam; -- 12.45
STDDEV_SAMP(value)	DOUBLE	计算value的**样本标准差**（Sample Standard Deviation）。参数须为数值类型；公式为 $\sqrt{(\Sigma(x-\mu)^2/(N-1))}$；跳过NULL；当输入行数 < 2 时返回NULL。	SELECT STDDEV_SAMP(score) FROM exam; -- 13.20
VAR_POP(value)	DOUBLE	计算value的**总体方差**（Population Variance）。参数须为数值类型；公式为 $\Sigma(x-\mu)^2/N$；跳过NULL；当输入行数 < 1 时返回NULL。	SELECT VAR_POP(score) FROM exam; -- 155.00
VAR_SAMP(value)	DOUBLE	计算value的**样本方差**（Sample Variance）。参数须为数值类型；公式为 $\Sigma(x-\mu)^2/(N-1)$；跳过NULL；当输入行数 < 2 时返回NULL。	SELECT VAR_SAMP(score) FROM exam; -- 174.38
COLLECT(value)	MULTISET	将value收集为多值集合（MULTISET），保留重复元素。参数可为任意类型；跳过NULL；返回类型为MULTISET<参数类型>。	SELECT COLLECT(tag) FROM articles; -- [a, b, b, c]

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
COLLECT(DISTINCT value)	MULTISET	将value去NULL并去重后收集为多值集合。参数可为任意类型；返回类型为MULTISET<参数类型>。	SELECT COLLECT(DISTINCT tag) FROM articles; -- [a, b, c]
LISTAGG(value, separator)	STRING	将value以separator为分隔符拼接为单个字符串。value须为字符串类型，separator须为字符串常量；跳过NULL；无分隔符时separator传空字符串 ''。	SELECT LISTAGG(name, ',') FROM users; -- Alice,Bob,Carol
GROUPING(expression)	INT	判断expression是否因GROUP BY ROLLUP或GROUP BY CUBE而被聚合填充为NULL。返回 1 表示该列被聚合（值由 ROLLUP/CUBE 产生），返回0表示该列保留了原始分组值。须在包含 ROLLUP/CUBE 的GROUP BY中使用。	SELECT year, region, GROUPING(region) AS g FROM sales GROUP BY ROLLUP (year, region); -- g=0（原始行）或 g=1（小计行）
BIT_AND(value)	INT / BIGINT	对value执行按位与聚合。参数须为整数类型（TINYINT / SMALLINT / INT / BIGINT）；等价于对所有非NULL值依次做&运算；跳过 NULL；仅一个非NULL值时返回该值本身。	SELECT BIT_AND(flags) FROM events; -- 8
BIT_OR(value)	INT / BIGINT	对value执行按位或聚合。参数须为整数类型（TINYINT / SMALLINT / INT / BIGINT）；等价于对所有非NULL值依次做|运算；跳过NULL；仅一个非NULL值时返回该值本身。	SELECT BIT_OR(flags) FROM events; -- 15

版权所有 © 华为云计算技术有限公司

6.9 JSON 函数

概述

JSON函数用于对JSON格式的字符串进行解析、查询、构造和验证操作。AI DataLake支持SQL标准（ISO/IEC TR 19075-6）中定义的JSON路径表达式，支持lax和strict两种模式，涵盖JSON有效性判断、路径查询、标量提取、JSON对象与数组构造等完整的JSON处理能力。

更多内容请参考**JSON Functions**。

适用场景

- **JSON数据验证**：使用IS JSON判断字符串是否为合法JSON，或是否为特定JSON类型。
- **JSON路径查询**：使用JSON_EXISTS判断路径是否存在，使用JSON_VALUE提取标量值，使用JSON_QUERY提取JSON片段。
- **JSON序列化**：使用JSON_STRING将值序列化为JSON字符串。
- **JSON构造**：使用JSON_OBJECT和JSON_ARRAY动态构建JSON对象和数组。

约束与限制

- JSON路径表达式使用符号表示根节点，支持属性访问（符号表示根节点，支持属性访问（.a）、数组元素访问（.a[0].b）和数组遍历（.*a*[0].*b*）和数组遍历（.a[*].b）。
- 路径模式分为lax和strict两种：lax模式对错误更宽容，将错误转为空序列；strict模式从模式角度检查数据，遇到不符合路径表达式的数据会抛出错误。省略时默认为strict模式。
- JSON_VALUE仅能提取标量值，非标量值无法返回；支持VARCHAR/STRING、BOOLEAN、INTEGER、DOUBLE四种返回类型。
- JSON_OBJECT的键必须为非NULL的字符串字面量，值可以为任意表达式。
- JSON_ARRAY省略ON NULL时默认为ABSENT ON NULL（忽略NULL值），而JSON_OBJECT省略ON NULL时默认为NULL ON NULL（将NULL作为null值包含）。
- Lax模式的部分功能当前未完全支持（上游缺陷CALCITE-4717），非标准行为不保证。

版权所有 © 华为云计算技术有限公司

语法结构

语法	返回类型	说明	示例
IS JSON [{ VALUE | SCALAR | ARRAY | OBJECT }]	BOOLEAN	判断给定字符串是否为合法JSON。 指定可选的type参数可约束允许的JSON对象类型。如果字符串是合法JSON但不是指定类型，则返回false。默认为VALUE。	-- TRUE '1' IS JSON '[]' IS JSON '{}' IS JSON -- TRUE '"abc"' IS JSON -- FALSE 'abc' IS JSON NULL IS JSON -- TRUE '1' IS JSON SCALAR -- FALSE '1' IS JSON ARRAY -- FALSE '1' IS JSON OBJECT -- FALSE '{}' IS JSON SCALAR -- FALSE '{}' IS JSON ARRAY -- TRUE '{}' IS JSON OBJECT
JSON_EXISTS(jsonValue, path [{ TRUE | FALSE | UNKNOWN | ERROR } ON ERROR])	BOOLEAN	判断JSON字符串是否满足给定的路径搜索条件。 如果省略错误处理行为，默认为FALSE ON ERROR。	-- TRUE SELECT JSON_EXISTS('{"a": true}', '$.a'); -- FALSE SELECT JSON_EXISTS('{"a": true}', '$.b'); -- TRUE SELECT JSON_EXISTS('{"a": [{ "b": 1 }]}', '$.a[0].b'); -- TRUE SELECT JSON_EXISTS('{"a": true}', 'strict $.b' TRUE ON ERROR); -- FALSE SELECT JSON_EXISTS('{"a": true}', 'strict $.b' FALSE ON ERROR);
JSON_STRING(value)	STRING	将值序列化为JSON。 该函数返回包含序列化值的JSON字符串。如果值为NULL，则函数返回NULL。	-- NULL JSON_STRING(CAST(NULL AS INT)) -- '1' JSON_STRING(1) -- 'true' JSON_STRING(TRUE) -- '"Hello, World!"' JSON_STRING('Hello, World!') -- '[1,2]' JSON_STRING(ARRAY[1, 2])

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
JSON_VALUE(jsonValue, path [RETURNING] [{ NULL | ERROR | DEFAULT } ON EMPTY] [{ NULL | ERROR | DEFAULT } ON ERROR])	STRING / BOOLEAN / INTEGER / DOUBLE	从JSON字符串中提取标量值。 该方法在JSON字符串中搜索给定的路径表达式，如果路径处的值为标量则返回该值。非标量值无法返回。默认情况下，值以STRING类型返回。使用dataType可选择不同的类型，支持的类型包括：VARCHAR / STRING、BOOLEAN、INTEGER、DOUBLE。 对于空路径表达式或错误，可以定义行为来返回null、抛出错误或返回定义的默认值。省略时默认分别为NULL ON EMPTY或NULL ON ERROR。默认值可以是字面量或表达式。如果默认值本身抛出错误，则回退到ON EMPTY的错误行为，并对ON ERROR抛出错误。 对于路径中包含空格等特殊字符的情况，可以使用['property']或["property"]来选择父对象中的指定属性。务必在属性名两侧加上单引号或双引号。在SQL中使用JSON_VALUE时，路径是一个已加单引号的字符参数，因此需要转义属性名周围的单引号，例如JSON_VALUE('{"a b": "true"}', '$.["a b"]')。	-- "true" JSON_VALUE('{"a": true}', '$.a') -- TRUE JSON_VALUE('{"a": true}', '$.a' RETURNING BOOLEAN) -- "false" JSON_VALUE('{"a": true}', 'lax $.b' DEFAULT FALSE ON EMPTY) -- "false" JSON_VALUE('{"a": true}', 'strict $.b' DEFAULT FALSE ON ERROR) -- 0.998D JSON_VALUE('{"a.b": [0.998,0.996]}','$.["a.b"][0]' RETURNING DOUBLE) -- "right" JSON_VALUE('{"contains blank": "right"}', 'strict $.["contains blank"]' NULL ON EMPTY DEFAULT 'wrong' ON ERROR)

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
JSON_QUERY(jsonValue, path [RETURNING] [{ WITHOUT | WITH CONDITIONAL | WITH UNCONDITIONAL } [ARRAY] WRAPPER] [{ NULL | EMPTY ARRAY | EMPTY OBJECT | ERROR } ON EMPTY] [{ NULL | EMPTY ARRAY | EMPTY OBJECT | ERROR } ON ERROR])	STRING / ARRAY<STRING>	从JSON字符串中提取JSON值。 结果以STRING或ARRAY<STRING>类型返回。可以通过RETURNING子句控制。 wrappingBehavior决定提取的值是否应包装为数组，以及是无条件包装还是仅当值本身不是数组时才包装。 onEmpty和onError分别决定路径表达式为空或发生错误时的行为。默认情况下，两种情况都返回null。其他选项包括使用空数组、空对象或抛出错误。	-- '{ "b": 1 }' JSON_QUERY('{ "a": { "b": 1 } }', '$.a') -- '[1, 2]' JSON_QUERY('[1, 2]', '$') -- NULL JSON_QUERY(CAST(NULL AS STRING), '$') -- '["c1","c2"]' JSON_QUERY('{"a":[{"c":"c1"}, {"c":"c2"}]}', 'lax $.a[*].c') -- 将结果包装到数组中 -- '[{}]' JSON_QUERY('{}', '$' WITH CONDITIONAL ARRAY WRAPPER) -- '[1, 2]' JSON_QUERY('[1, 2]', '$' WITH CONDITIONAL ARRAY WRAPPER) -- '[[1, 2]]' JSON_QUERY('[1, 2]', '$' WITH UNCONDITIONAL ARRAY WRAPPER) -- 必须包装标量才能返回 -- NULL JSON_QUERY(1, '$') -- '[1]' JSON_QUERY(1, '$' WITH CONDITIONAL ARRAY WRAPPER) -- 路径表达式为空/存在错误时的行为 -- '{}' JSON_QUERY('{}', 'lax $.invalid' EMPTY OBJECT ON EMPTY) -- '[]' JSON_QUERY('{}', 'strict $.invalid' EMPTY ARRAY ON ERROR)
JSON_OBJECT([[KEY] key VALUE value]* [{ NULL | ABSENT } ON NULL])	STRING	从键值对列表构建JSON对象字符串。 注意键必须为非NULL的字符串字面量，而值可以为任意表达式。 该函数返回JSON字符串。ON NULL行为定义如何处理NULL值。省略时默认为NULL ON NULL。 从其他JSON构造函数调用（JSON_OBJECT、JSON_ARRAY）创建的值会直接插入而非作为字符串插入。这允许构建嵌套的JSON结构。	-- '{}' JSON_OBJECT() -- '{"K1":"V1","K2":"V2"}' JSON_OBJECT('K1' VALUE 'V1', 'K2' VALUE 'V2') -- Expressions as values JSON_OBJECT('orderNo' VALUE orders.orderId) -- ON NULL JSON_OBJECT(KEY 'K1' VALUE CAST(NULL AS STRING) NULL ON NULL) -- '{"K1":null}' JSON_OBJECT(KEY 'K1' VALUE CAST(NULL AS STRING) ABSENT ON NULL) -- '{}' -- '{"K1":{"K2":"V"}}' JSON_OBJECT(KEY 'K1' VALUE JSON_OBJECT(KEY 'K2' VALUE 'V'))

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
JSON_ARRAY([value]* [{ NULL | ABSENT } ON NULL])	STRING	从值列表构建JSON数组字符串。 该函数返回JSON字符串。值可以为任意表达式。ON NULL行为定义如何处理NULL值。省略时默认为ABSENT ON NULL。 从其他JSON构造函数调用（JSON_OBJECT、JSON_ARRAY）创建的元素会直接插入而非作为字符串插入。这允许构建嵌套的JSON结构。	-- '[]' JSON_ARRAY() -- '[1,"2"]' JSON_ARRAY(1, '2') -- Expressions as values JSON_ARRAY(orders.orderId) -- ON NULL JSON_ARRAY(CAST(NULL AS STRING) NULL ON NULL) -- '[null]' JSON_ARRAY(CAST(NULL AS STRING) ABSENT ON NULL) -- '[]' -- '[[1]]' JSON_ARRAY(JSON_ARRAY(1))

6.10 值构造函数

概述

值构造函数用于构造复合数据类型的值，包括行（ROW）、数组（ARRAY）和映射（MAP）。AI DataLake支持SQL标准中的隐式和显式行构造、数组构造和映射构造能力。

更多内容请参考**Value Construction Functions**。

适用场景

- **复合行构造**：使用ROW构造函数创建包含多个字段的行值。
- **数组构造**：使用ARRAY构造函数创建数组值，用于IN子句、UNNEST操作等。
- **映射构造**：使用MAP构造函数创建键值对映射，用于配置参数、属性存储等。
- **窗口定义**：使用行数间隔构造函数辅助窗口定义。

约束与限制

- 隐式行构造（使用圆括号）要求至少两个字段，显式ROW构造函数可接受任意数量的字段。
- ARRAY和MAP构造函数中的值类型应保持一致或兼容。
- MAP构造函数的参数必须成对出现，即键值对数量必须为偶数。
- 行数间隔构造函数（N rows）仅Table API支持，SQL中不可用。

版权所有 © 华为云计算技术有限公司

语法结构

语法	返回类型	说明	示例
implicit constructor with parenthesis (value1 [, value2]*) **explicit** ROW constructor with ROW(value1 [, value2]*)	ROW	从值列表（value1, value2,…）创建行。 隐式行构造器要求至少两个字段。显式行构造器可处理任意数量的字段。两者均支持任意表达式作为字段。	-- 隐式行构造器（至少两个字段） SELECT (1, 'hello'); -- 显式行构造器（可处理任意数量字段） SELECT ROW(1); SELECT ROW(1, 'hello', TRUE); -- 使用表达式作为字段 SELECT ROW(orderId, orderAmount) FROM orders;
ARRAY ‘[’ value1 [, value2]* ‘]’	ARRAY	从值列表（value1, value2, …）创建数组。	-- 从字面量创建数组 SELECT ARRAY [1, 2, 3]; -- 从表达式创建数组 SELECT ARRAY [orderId, orderAmount] FROM orders; -- 嵌套数组 SELECT ARRAY [ARRAY [1, 2], ARRAY [3, 4]];
MAP ‘[’ value1, value2 [, value3, value4]* ‘]’	MAP	从键值对列表（(value1, value2), (value3, value4), …）创建映射。	-- 从字面量创建映射 SELECT MAP ['a', 1, 'b', 2]; -- 从表达式创建映射 SELECT MAP ['orderId', orderId, 'amount', orderAmount] FROM orders;

6.11 值访问函数

概述

值访问函数用于访问复合数据类型（如Tuple、POJO等）中的字段值，支持按名称访问单个字段和展开所有子字段。AI DataLake支持SQL标准中通过点号语法访问复合类型字段的能力。

更多内容请参考**Value Access Functions**。

适用场景

- **字段访问**：从复合类型（如Tuple、POJO）中按名称获取特定字段的值。
- **字段展开**：将复合类型的所有直接子类型展开为独立字段，用于SELECT查询中展开嵌套结构。

版权所有 © 华为云计算技术有限公司

约束与限制

- 点号访问语法（tableName.compositeType.field）仅适用于Flink复合类型（Tuple、POJO等），不适用于普通标量字段。
- 展开操作（tableName.compositeType.*）将每个直接子类型转换为独立字段，字段命名通常与原始字段类似，但使用美元符号分隔（如mypojo$mytuple$f0）。
- 展开操作仅展开直接子类型，不会递归展开嵌套的复合类型。

语法结构

语法	返回类型	说明	示例
tableName.compositeType.field	ANY	按名称返回Flink复合类型（如Tuple、POJO）中字段的值。	-- 假设表myTable包含ROW类型字段 myRow(field1 INT, field2 STRING) -- 按名称访问field1字段 SELECT myTable.myRow.field1 FROM myTable; -- 按名称访问嵌套结构中的字段 -- 假设person为ROW(name STRING, age INT) SELECT myTable.person.name FROM myTable;
tableName.compositeType.*	ANY	返回Flink复合类型（如Tuple、POJO）的扁平化表示，将每个直接子类型转换为独立字段。 在大多数情况下，扁平化表示的字段名称与原始字段类似，但使用美元符号作为分隔符（如mypojo$mytuple$f0）。	-- 假设表myTable包含ROW类型字段 myRow(field1 INT, field2 STRING) -- 扁平化展开myRow的所有字段，展开后字段名称使用美元符号分隔符（如mypojo$mytuple$f0） SELECT myTable.myRow.* FROM myTable;

6.12 分组函数

概述

分组函数用于GROUP BY扩展（如ROLLUP、CUBE、GROUPING SETS）中标识分组级别。AI DataLake支持SQL标准中的GROUP_ID和GROUPING/GROUPING_ID函数，用于区分不同分组组合并判断列是否参与当前分组聚合。

更多内容请参考**Grouping Functions**。

适用场景

- **多级聚合报表**：在ROLLUP、CUBE、GROUPING SETS等多级分组聚合中标识不同的分组组合。
- **小计与合计区分**：通过GROUPING函数判断某列是否在当前分组中被聚合（返回NULL），从而区分原始数据行和聚合行。
- **条件过滤**：在HAVING子句中使用GROUPING结果过滤特定分组级别。

版权所有 © 华为云计算技术有限公司

约束与限制

- GROUP_ID和GROUPING/GROUPING_ID函数仅在包含ROLLUP、CUBE或GROUPING SETS的GROUP BY语句中有意义。
- GROUPING_ID是GROUPING的别名，两者语义等价。
- GROUPING返回的位向量中，0表示对应列参与了GROUP BY，1表示对应列未参与（被聚合为NULL）。

语法结构

语法	返回类型	说明	示例
GROUP_ID()	INTEGER	返回唯一标识分组键组合的整数。	-- 配合ROLLUP使用，标识每个分组层级 SELECT dept, role, GROUP_ID() AS gid, COUNT(*) AS cnt FROM employees GROUP BY ROLLUP(dept, role); -- 结果示例： -- dept=Sales, role=Manager, gid=0, cnt=5 -- dept=Sales, role=NULL, gid=0, cnt=10 -- dept=NULL, role=NULL, gid=0, cnt=50
GROUPING(expression1 [, expression2]*) GROUPING_ID(expression1 [, expression2]*)	INTEGER	返回给定分组表达式的位向量。	-- 配合ROLLUP使用，判断分组键是否被聚合 SELECT dept, role, GROUPING(dept, role) AS grp_bits, COUNT(*) AS cnt FROM employees GROUP BY ROLLUP(dept, role); -- 结果示例： -- dept=Sales, role=Manager, grp_bits=0, cnt=5 -- dept=Sales, role=NULL, grp_bits=1, cnt=10 -- dept=NULL, role=NULL, grp_bits=3, cnt=50 -- 使用CASE WHEN区分聚合层级 SELECT CASE WHEN GROUPING(dept, role) = 0 THEN '明细' WHEN GROUPING(dept, role) = 1 THEN '部门小计' WHEN GROUPING(dept, role) = 3 THEN '总计' END AS level, dept, role, COUNT(*) AS cnt FROM employees GROUP BY ROLLUP(dept, role);

6.13 哈希函数

概述

哈希函数用于计算字符串的散列值，支持MD5、SHA-1、SHA-224、SHA-256、SHA-384、SHA-512等常用哈希算法。AI DataLake支持SQL标准中的哈希计算能力，返回十六进制字符串形式的散列值。

更多内容请参考**Hash Functions**。

适用场景

- **数据校验**：使用MD5或SHA哈希对数据进行完整性校验。
- **敏感信息脱敏**：对敏感字段（如身份证号、手机号）进行哈希脱敏处理。
- **数据分区**：使用哈希值进行数据分片或分布式分区键生成。

版权所有 © 华为云计算技术有限公司

- **指纹比对**：通过哈希值快速判断数据是否一致。

约束与限制

- 所有哈希函数的输入参数必须为STRING类型，若输入为NULL则返回NULL。
- MD5返回32位十六进制字符串，SHA-1返回40位，SHA-224返回56位，SHA-256返回64位，SHA-384返回96位，SHA-512返回128位。
- SHA2函数的hashLength参数仅支持224、256、384、512四种值，其他值会导致错误。
- 哈希函数为单向函数，不可逆，不适用于加密场景。

语法结构

语法	返回类型	说明	示例
MD5(string)	STRING	返回string的MD5哈希值，为32位十六进制数字字符串；如果string为NULL则返回NULL。	-- 返回 '5d41402abc4b2a76b9719d911017c592' SELECT MD5('hello'); -- 对表字段计算哈希值 SELECT userId, MD5(userEmail) AS emailHash FROM users; -- 输入为NULL时返回NULL SELECT MD5(CAST(NULL AS STRING));
SHA1(string)	STRING	返回string的SHA-1哈希值，为40位十六进制数字字符串；如果string为NULL则返回NULL。	-- 返回 'aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d' SELECT SHA1('hello'); -- 对表字段计算哈希值 SELECT orderId, SHA1(orderDetail) AS detailHash FROM orders;
SHA224(string)	STRING	返回string的SHA-224哈希值，为56位十六进制数字字符串；如果string为NULL则返回NULL。	-- 返回56位十六进制数字字符串 SELECT SHA224('hello'); -- 对表字段计算哈希值 SELECT userId, SHA224(userName) AS nameHash FROM users;
SHA256(string)	STRING	返回string的SHA-256哈希值，为64位十六进制数字字符串；如果string为NULL则返回NULL。	-- 返回 '2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824' SELECT SHA256('hello'); -- 对表字段计算哈希值 SELECT userId, SHA256(password) AS passwordHash FROM users; -- 输入为NULL时返回NULL SELECT SHA256(CAST(NULL AS STRING));
SHA384(string)	STRING	返回string的SHA-384哈希值，为96位十六进制数字字符串；如果string为NULL则返回NULL。	-- 返回96位十六进制数字字符串 SELECT SHA384('hello'); -- 对表字段计算哈希值 SELECT docId, SHA384(docContent) AS contentHash FROM documents;

版权所有 © 华为云计算技术有限公司

语法	返回类型	说明	示例
SHA512(string)	STRING	返回string的SHA-512哈希值，为128位十六进制数字字符串；如果string为NULL则返回NULL。	-- 返回128位十六进制数字字符串 SELECT SHA512('hello'); -- 对表字段计算哈希值 SELECT fileId, SHA512(fileContent) AS contentHash FROM files;
SHA2(string, hashLength)	STRING	使用SHA-2系列哈希函数（SHA-224、SHA-256、SHA-384或SHA-512）返回哈希值。第一个参数string是要哈希的字符串，第二个参数hashLength是结果的位长度（224、256、384或512）。如果string或hashLength为NULL则返回NULL。	-- 使用SHA-256（等价于SHA256函数）返回 '2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824' SELECT SHA2('hello', 256); -- 使用SHA-512（等价于SHA512函数） SELECT SHA2('hello', 512); -- 使用SHA-224（等价于SHA224函数） SELECT SHA2('hello', 224); -- hashLength为NULL时返回NULL SELECT SHA2('hello', CAST(NULL AS INT));

版权所有 © 华为云计算技术有限公司

7 辅助语句

7.1 USE 语句

概述

USE语句用于切换当前会话的默认Catalog。切换后，后续SQL中引用表时可以省略Catalog名称前缀，直接使用表名即可。

适用场景

- **多Catalog操作**：在HiveCatalog和默认Catalog之间切换，访问不同元数据源。
- **多租户场景**：不同数据库隔离不同业务的数据，通过USE快速切换工作上下文。

约束与限制

- USE CATALOG要求目标Catalog已通过CREATE CATALOG注册。
- USE仅影响当前会话，不影响其他会话或全局配置。

语法结构

```
-- 切换当前Catalog
USE CATALOG catalog_name;
```

使用示例

示例：USE CATALOG切换Catalog

```
CREATE TABLE test_kafka (
  user_id varchar,
  item_id varchar,
  cat_id varchar,
  test_time timestamp
) WITH (
  'properties.bootstrap.servers' = 'Kafka的Broker实例业务IP:Kafka端口号',
  'format' = 'json',
  'topic' = 'zw_test_kafka',
  'connector' = 'kafka',
  'scan.startup.mode' = 'latest-offset'
```

 版权所有 © 华为云计算技术有限公司

```
);
CREATE CATALOG myhive
WITH
  (
    'type' = 'hive',
    'metastore' = 'lakeformation'
  );
use catalog myhive;
CREATE TABLE IF NOT EXISTS `default`.user_behavior_hive_tbl (
    user_id STRING,
    item_id STRING,
    cat_id STRING,
    ts timestamp,
    dy STRING,
    ho STRING,
    mi STRING
  ) PARTITIONED BY (dy, ho, mi) with (
    'connector' = 'hive',
    'partition.time-extractor.timestamp-pattern' = '$dy $ho:$mi:00',
    'sink.partition-commit.trigger' = 'process-time',
    'sink.partition-commit.delay' = '0S',
    'sink.partition-commit.policy.kind' = 'metastore,success-file',
    'hive.storage.file-format'='parquet'
);

INSERT into
  `default`.user_behavior_hive_tbl
SELECT
  user_id,
  item_id,
  cat_id,
  test_time,
  DATE_FORMAT(test_time, 'yyyy-MM-dd'),
  DATE_FORMAT(test_time, 'HH'),
  DATE_FORMAT(test_time, 'mm')
FROM
  default_catalog.default_database.test_kafka;
```

7.2 SET 语句

概述

SET是Flink SQL的会话配置管理语句。SET用于设置指定配置项的值。在AI DataLake中，通过SET可动态调整作业运行参数，无需重启作业即可影响后续SQL的执行行为。

适用场景

- **调整并行度**：通过SET修改parallelism.default，控制作业的并行执行度。
- **切换运行模式**：通过SET修改execution.runtime-mode，在流模式和批模式间切换。
- **调优状态TTL**：通过SET修改table.exec.state.ttl，控制状态保留时长以平衡性能与正确性。
- **配置检查点**：通过SET修改execution.checkpointing.*系列参数，控制检查点行为。
- **排查问题**：通过SET查看当前所有配置，确认参数是否生效。

约束与限制

- SET仅影响当前会话，不会持久化到全局配置。

 版权所有 © 华为云计算技术有限公司

- 部分配置项修改后仅对后续提交的作业生效，已运行中的作业不受影响。
- SET设置的值在会话结束后自动失效。
- 部分配置项在AI DataLake平台上的支持范围可能与开源Flink不同，请以实际环境为准。

语法结构

修改配置

```
SET ('key' = 'value')
```

常用配置参数

参数	默认值	说明
parallelism.default	1	作业默认并行度。
execution.runtime-mode	STREAMING	运行模式，可选STREAMING或BATCH。
table.exec.state.ttl	0 ms	状态生存时间，超过该时间未访问的状态将被清理。
table.exec.source.idle-timeout	0 ms	Source空闲超时时间，空闲标记用于Watermark推进。
execution.checkpointing.interval	无	检查点触发间隔。
execution.checkpointing.mode	EXACTLY_ONCE	检查点模式，可选EXACTLY_ONCE或AT_LEAST_ONCE。
execution.checkpointing.timeout	10 min	检查点超时时间。
state.backend	hashmap	状态后端类型，可选hashmap或rocksdb。

使用示例

示例1：查看和设置并行度

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
id BIGINT,
name STRING,
amount DECIMAL(10, 2)
) WITH (
'connector' = 'datagen',
'rows-per-second' = '100',
'fields.id.kind' = 'sequence',
'fields.id.start' = '1',
'fields.id.end' = '10000',
'fields.name.length' = '8'
);
-- 创建Print结果表
```

 版权所有 © 华为云计算技术有限公司

```
CREATE TABLE print_sink (
id BIGINT,
name STRING,
amount DECIMAL(10, 2)
) WITH (
'connector' = 'print',
'print-identifier' = 'set_demo'
);
-- 设置并行度为4
SET 'parallelism.default' = '4';
-- 提交作业
INSERT INTO print_sink
SELECT id, name, amount
FROM datagen_source;
```

示例2：设置检查点和运行模式

```
-- 创建DataGen源表
CREATE TABLE datagen_source (
order_id BIGINT,
customer_id INT,
order_time TIMESTAMP(3),
WATERMARK FOR order_time AS order_time - INTERVAL '5' SECOND
) WITH (
'connector' = 'datagen',
'rows-per-second' = '50',
'fields.order_id.kind' = 'sequence',
'fields.order_id.start' = '1',
'fields.order_id.end' = '50000',
'fields.customer_id.min' = '1',
'fields.customer_id.max' = '1000'
);
-- 创建Print结果表
CREATE TABLE print_sink (
order_id BIGINT,
customer_id INT
) WITH (
'connector' = 'print',
'print-identifier' = 'checkpoint_demo'
);
-- 设置检查点间隔为30秒
SET 'execution.checkpointing.interval' = '30s';
-- 设置检查点模式为EXACTLY_ONCE
SET 'execution.checkpointing.mode' = 'EXACTLY_ONCE';
-- 设置状态TTL为1小时
SET 'table.exec.state.ttl' = '3600000 ms';
-- 提交作业
INSERT INTO print_sink
SELECT order_id, customer_id
FROM datagen_source;
```

 版权所有 © 华为云计算技术有限公司

8 Flink 增强特性使用介绍

8.1 增强函数说明

8.1.1 BITAND

语法

```
INT BITAND(INT number1,INT number2)
```

描述

BITAND函数用于执行按位与操作。按位与操作是对两个整数的二进制位逐位进行比较，当两个对应位均为1时，结果位为1，否则为0。

入参

参数	数据类型
number1	INT
number2	INT

说明

输入和输出类型均为整型，且类型一致。

示例

```
CREATE TABLE source (
  f_sequence INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE print with(
'connector' = 'print'
```


版权所有 © 华为云计算技术有限公司

```
) as select
  BITAND (2, 3) as int1
from source;
```

返回结果为2，数据类型为INT。

8.1.2 BITNOT

语法

```
INT BITNOT(INT number)
```

描述

BITNOT函数用于执行按位取反操作。按位取反操作是将整数的二进制位逐位取反，即0变1、1变0。由于INT类型采用补码表示，取反后的结果需结合补码规则理解。

入参

参数	数据类型
number	INT

示例

```
CREATE TABLE source (
  f_sequence INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE print with(
'connector' = 'print'
) as select
  BITNOT (7) as int1
from source;
```

返回结果为-8，数据类型为INT。

8.1.3 BITOR

语法

```
INT BITOR(INT number1, INT number2)
```

描述

BITOR函数用于执行按位或操作。按位或操作是对两个整数的二进制位逐位进行比较，当两个对应位中至少有一个为1时，结果位为1，否则为0。

入参

参数	数据类型
number1	INT

版权所有 © 华为云计算技术有限公司

参数	数据类型
number2	INT

说明

输入和输出类型均为整型，且类型一致。

示例

```
CREATE TABLE source (
  f_sequence INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE print with(
'connector' = 'print'
) as select
  BITOR (2, 3) as int1
from source;
```

返回结果为3，数据类型为INT。

8.1.4 BITXOR

语法

```
INT BITXOR(INT number1, INT number2)
```

描述

BITXOR函数用于执行按位异或操作。按位异或操作是对两个整数的二进制位逐位进行比较，当两个对应位不同时结果位为1，相同时为0。

入参

参数	数据类型
number1	INT
number2	INT

说明

输入和输出类型均为整型，且类型一致。

示例

```
CREATE TABLE source (
  f_sequence INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
```


版权所有 © 华为云计算技术有限公司

```
);
CREATE TABLE print with(
'connector' = 'print'
) as select
  BITXOR (2, 3) as int1
from source;
```

返回结果为1，数据类型为INT。

8.1.5 DATE_ADD

语法

```
STRING DATE_ADD(VARCHAR startdate, INT days)
STRING DATE_ADD(TIMESTAMP time, INT days)
```

描述

DATE_ADD函数用于返回指定日期增加目标天数后的日期。当目标天数为负数时，相当于减去对应天数。

参数说明

- 指定日期：数据类型为TIMESTAMP或者STRING（格式为：yyyy-MM-dd HH:mm:ss），支持参数为NULL的特殊情况。
- 目标天数：数据类型为INT。
- 返回日期：指定日期增加目标天数后的日期，数据类型为STRING。

示例

```
CREATE TABLE source (
  time1 TIMESTAMP
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE Sink (
  date1 string,
  date2 string,
  date3 string
) with ('connector' = 'print');
INSERT INTO
  Sink
SELECT
  DATE_ADD(time1, 30) as date1,
  DATE_ADD('2017-09-15 00:00:00', 30) as date2,
  DATE_ADD(cast(null as timestamp),30) as date3
FROM source
```

表 8-1 返回结果

date1 (string)	date2 (string)	date3 (string)
2024-06-28	2017-10-15	null

版权所有 © 华为云计算技术有限公司

8.1.6 DATE_SUB

语法

```
STRING DATE_SUB(STRING startdate, INT days)
STRING DATE_SUB(TIMESTAMP time, INT days)
```

描述

DATE_SUB函数用于返回指定日期减去目标天数后的日期。当目标天数为负数时，相当于增加对应天数。

参数说明

- 指定日期：数据类型为TIMESTAMP或者STRING（格式为：yyyy-MM-dd HH:mm:ss），支持参数为NULL的特殊情况。
- 目标天数：数据类型为INT。
- 返回日期：指定日期减去目标天数后的日期，数据类型为STRING。

示例

```
CREATE TABLE source (
  time1 TIMESTAMP
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE Sink (
  date1 string,
  date2 string,
  date3 string
) with ('connector' = 'print');
INSERT into
  Sink
select
  DATE_SUB(time1,30) as date1,
  DATE_SUB('2017-09-15 00:00:00', 30) as date2,
  DATE_SUB(cast(null as timestamp),30) as date3
FROM source
```

表 8-2 返回结果

date1 (string)	date2 (string)	date3 (string)
2024-04-29	2017-08-16	null

8.1.7 DATEDIFF

语法

```
INT DATEDIFF(VARCHAR enddate, VARCHAR startdate)
INT DATEDIFF(TIMESTAMP enddate, VARCHAR startdate)
INT DATEDIFF(VARCHAR enddate, TIMESTAMP startdate)
INT DATEDIFF(TIMESTAMP enddate, TIMESTAMP startdate)
```


版权所有 © 华为云计算技术有限公司

描述

DATEDIFF函数用于计算两个日期之间的天数差值。计算方式为enddate减去startdate，即返回值 = enddate - startdate。当enddate晚于startdate时返回正数，当enddate早于startdate时返回负数。

入参

参数	数据类型
enddate	TIMESTAMP、VARCHAR
startdate	TIMESTAMP、VARCHAR

说明

- VARCHAR日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。
- 如果入参中任何一个为NULL，则返回为NULL。

示例

```
CREATE TABLE source (
  time1 TIMESTAMP
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE print with(
'connector' = 'print'
) as select
DATEDIFF('2025-10-15 00:00:00', '2025-09-15 00:00:00') as int1,
        DATEDIFF(time1 ,'2025-12-01 00:00:00') as int2,
        DATEDIFF('2025-12-01',time1 ) as int3,
        DATEDIFF(time1 ,CAST(null as VARCHAR)) as int4,
        DATEDIFF(CAST(null as VARCHAR), '2025-12-01 00:00:00') as int5,
        DATEDIFF(CAST(null as VARCHAR), time1 ) as int6,
        DATEDIFF(time1 ,time1 ) as int7
from source;
```

表 8-3 返回结果

int1(INT)	int2(INT)	int3(INT)	int4(INT)	int5(INT)	int6(INT)	int7(INT)
30	49	-49	null	null	null	0

8.1.8 URL_DECODE

语法

```
VARCHAR URL_DECODE(VARCHAR input)
VARCHAR URL_DECODE(VARCHAR input, VARCHAR encoding)
```


版权所有 © 华为云计算技术有限公司

描述

URL_DECODE函数用于将application/x-www-form-urlencoded MIME格式的输入字符串转换为标准字符串。

入参

参数	数据类型	说明
input	VARCHAR	该参数指定要输入的字符串。
encoding	VARCHAR	可选参数，此参数用于指定解码格式，可以是GBK或者UTF-8等，如果不指定此参数则使用默认值UTF-8。

示例

```
CREATE TABLE source (
  f_sequence INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE print with(
'connector' = 'print'
) as select
  URL_DECODE ('http://calcite.apache.org') as value1,
  URL_DECODE ('https%3A%2F%2Fcalcite.apache.org') as value2,
  URL_DECODE ('http%3A%2F%2Ftest%3Fa%3Db%26c%3Dd') as value3,
  URL_DECODE ('http%3A%2F%2F%E4%BD%A0%E5%A5%BD') as value4,
  URL_DECODE ('test') as value5,
  URL_DECODE ('https%%3A%2F%2Fcalcite.apache.org') as value6,
  URL_DECODE ('https%3A%2F%2Fcalcite.apache.org%','UTF-8') as value7,
  URL_DECODE ('https%3A%2F%2Fcalcite.apache.org', 'gb2312') as value8
from source;
```

表 8-4 返回结果

value1(VARCHAR)	value2(VARCHAR)	value3(VARCHAR)	value4(VARCHAR)	value5(VARCHAR)	value6(VARCHAR)	value7(VARCHAR)	value8(VARCHAR)
http://calcite.apache.org	http://calcite.apache.org	http://test?a=b&c=d	http://你好	test	null	null	https://calcite.apache.org

8.1.9 URL_ENCODE

语法

```
VARCHAR URL_ENCODE(VARCHAR input)
VARCHAR URL_ENCODE(VARCHAR input, VARCHAR encoding)
```


版权所有 © 华为云计算技术有限公司

描述

URL_ENCODE用于以application/x-www-form-urlencoded MIME格式对输入字符串进行编码并返回编码后的字符串。

入参

参数	数据类型	说明
input	VARCHAR	该参数指定要输入的字符串。
encoding	VARCHAR	可选参数，此参数用于指定编码格式，可以是GBK或者UTF-8等，如果不指定此参数则使用默认值UTF-8。

示例

```
CREATE TABLE source (
  f_sequence INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
CREATE TABLE print with(
'connector' = 'print'
) as select
  URL_ENCODE ('http://calcite.apache.org') as value1,
  URL_ENCODE ('http://test?a=b&c=d') as value2,
  URL_ENCODE ('http://你好') as value3,
  URL_ENCODE ('test') as value4,
  URL_ENCODE ('http://你好', 'gb2312') as value5
from source;
```

表 8-5 返回结果

value1(VARCHAR)	value2(VARCHAR)	value3(VARCHAR)	value4(VARCHAR)	value5(VARCHAR)
http%3A%2F%2Fcalcite.apache.org	http%3A%2F%2Ftest%3Fa%3Db%26c%3Dd	http%3A%2F%2F%E4%BD%A0%E5%A5%BD	test	http%3A%2F%2F%C4%E3%BA%C3

8.2 FlinkSQL 使用 RocksDB Merge Operator 提升 Join 算子性能

使用场景

在使用RocksDB状态后端的FlinkSQL作业中，包含NoUniqueKey的Join算子在更新状态时需要先读取旧值再写入新值，产生大量的读放大开销，导致Join算子性能下降，开启该特性后，Join算子状态更新可直接通过Merge操作完成，从而提升Join算子的计算性能。

版权所有 © 华为云计算技术有限公司

约束与限制

- 仅使用RocksDB状态后端时，该特性会提升Join算子的计算性能。
- 开启该特性前后生成的快照，互相不兼容。

使用方法

可通过以下两种方式配置table.exec.join.use-merge-operator为true，以开启MergeOperator功能：

- **方式一**：配置Flink作业时，在作业flink_config参数中添加该自定义参数。
- **方式二**：在端点的Flink参数中添加该自定义参数。

8.3 FlinkSQL 使用 RocksDB Prefix Bloom Filter 提升 Join 算子和 TopN 算子性能

使用场景

针对使用了Join算子、TopN算子的业务SQL，使用RocksDB Prefix Bloom Filter可以提升该类业务SQL的计算性能。

约束与限制

开启该特性前后生成的快照，互相不兼容。

使用方法

可通过以下两种方式配置table.exec.use-map-prefix-bloomfilter为true，以开启RocksDB Prefix Bloom Filter功能：

- **方式一**：配置Flink作业时，在作业flink_config参数中添加该自定义参数。
- **方式二**：在端点的Flink参数中添加该自定义参数。

可通过以下两种方式配置state.backend.rocksdb.auto-workload-parameters-derivation为true，以开启RocksDB自动负载参数推导功能：

- **方式一**：配置Flink作业时，在作业flink_config参数中添加该自定义参数。
- **方式二**：在端点的Flink参数中添加该自定义参数。

开启RocksDB自动负载参数推导功能后，可根据作业负载自动调整Prefix Bloom Filter相关参数。

8.4 FlinkSQL SPLIT_INDEX 函数性能优化

使用场景

使用FlinkSQL内置SPLIT_INDEX函数解析一个字符串的多个字段时，开启SPLIT_INDEX函数性能优化后，当解析同一个字符串的多个字段时，复用上次split的解析结果，提升计算性能。

版权所有 © 华为云计算技术有限公司

使用方法

可通过以下两种方式配置table.optimizing.split-index-optimization为true，以开启SPLIT_INDEX函数性能优化：

- **方式一**：配置Flink作业时，在作业flink_config参数中添加该自定义参数。
- **方式二**：在端点的Flink参数中添加该自定义参数。

SQL 示例

```
CREATE TABLE datagen (
  auction BIGINT,
  bidder BIGINT,
  price BIGINT,
  channel VARCHAR,
  url VARCHAR
)
WITH
  ('connector' = 'datagen');

CREATE TABLE blackhole (
  auction BIGINT,
  bidder BIGINT,
  price BIGINT,
  channel VARCHAR,
  dir1 VARCHAR,
  dir2 VARCHAR,
  dir3 VARCHAR
)
WITH
  ('connector' = 'blackhole');

INSERT INTO
  blackhole
SELECT
  auction,
  bidder,
  price,
  channel,
  SPLIT_INDEX (url, '/', 3) as dir1,
  SPLIT_INDEX (url, '/', 4) as dir2,
  SPLIT_INDEX (url, '/', 5) as dir3
FROM
  datagen;
```

8.5 FlinkSQL 使用 Object 复用特性提升 FlinkSQL 计算性能

使用场景

在FlinkSQL业务中，Table API开发的SQL业务在数据处理过程中会频繁创建和销毁对象（Object），导致大量的内存分配和垃圾回收（GC）开销，影响计算性能。开启Object复用特性后，FlinkSQL可在算子间复用数据传输对象，避免频繁的对象创建与销毁，从而提升SQL的计算性能。

约束与限制

- 该特性仅对Table API开发的SQL业务有效。
- 为保证计算准确性，FlinkSQL UDF中的Object无法复用。

版权所有 © 华为云计算技术有限公司

使用方法

可通过以下两种方式配置pipeline.serialization-elimination为true，以开启Object复用功能：

- **方式一**：配置Flink作业时，在作业flink_config参数中添加该自定义参数。
- **方式二**：在端点的Flink参数中添加该自定义参数。

8.6 FlinkSQL Lookup Join 支持 PROCTIME()语法

使用场景

Flink Lookup Join语法中要求FOR SYSTEM_TIME AS OF包含proc_time字段，例如FOR SYSTEM_TIME AS OF o.proc_time。现在可以使用PROCTIME()代替表中的proc_time字段，直接使用FOR SYSTEM_TIME AS OF PROCTIME()，表中不需要再定义proc_time字段。子查询也适用。

使用方法

配置Flink作业时，Lookup Join可以直接使用PROCTIME()来代替表中proc_time字段。

SQL 示例

```
CREATE TABLE kafkasource(
  order_id STRING,
  price DECIMAL(32, 2),
  currency STRING
) WITH (
  'connector' = 'kafka',
  'topic' = 'test_source',
  'properties.bootstrap.servers' = 'Kafka的Broker实例业务IP:Kafka端口号',
  'properties.group.id' = 'testGroup',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv'
);

CREATE CATALOG lf_paimon_catalog WITH (   'type' = 'paimon',   'metastore' = 'lakeformation' );
CREATE TABLE lf_paimon_catalog.paimon_db.paimon_table (id STRING, name STRING, country STRING);

CREATE TABLE print (
  order_id STRING,
  price DECIMAL(32, 2),
  name STRING,
  country STRING
) WITH ('connector' = 'print');
insert into
  print
SELECT
  o.order_id,
  o.price,
  c.name,
  c.country
FROM
  kafkasource as o
  JOIN lf_paimon_catalog.paimon_db.paimon_table FOR SYSTEM_TIME AS OF PROCTIME() AS c ON
o.order_id = c.id;
```


版权所有 © 华为云计算技术有限公司

8.7 FlinkSQL JSON_VALUE 函数性能优

使用场景

内置JSON_VALUE函数解析一个JSON item的多个字段时，复用上次JSON item的解析结果，提升算子性能。

使用方法

配置Flink作业时，可通过在FlinkServer WebUI的Flink作业开发界面添加自定义参数“table.optimizing.json-value-optimization”为“true”开启JSON_VALUE函数性能优化。

SQL 示例

```
CREATE TABLE kafkaSource (
    msgValue STRING
) with (
    'connector' = 'kafka'
    ,'topic' = 'topic-1'
    ,'properties.bootstrap.servers' = 'Kafka的Broker实例业务IP:Kafka端口号
    ,'properties.group.id' = 'group-1'
    ,'value.format' = 'raw'
    ,'scan.startup.mode' = 'group-offsets'
    ,'properties.auto.offset.reset' = 'earliest'
    ,'value.fields-include' = 'EXCEPT_KEY'
);
CREATE TABLE print (
    id STRING,
    name STRING
) with (
      'connector' = 'print'
);
INSERT INTO print
SELECT
    JSON_VAL(msgValue, '$.id'),
    JSON_VAL(msgValue, '$.name')
from kafkaSource;
```

8.8 FlinkSQL Kafka 和 upsert-kafka Connector 支持限流读

使用场景

使用FlinkSQL的Kafka、upsert-kafka Connector消费数据时需要限流。

使用方法

在创建的Source流表中添加“subtask.scan.records-per-second.limit”参数，该参数表示每秒消费Kafka单分区记录数，因此Source端整体限流速率为：min(source parallelism * subtask.scan.records-per-second.limit，kafka partitions num * subtask.scan.records-per-second.limit)。

版权所有 © 华为云计算技术有限公司

SQL 示例

```
CREATE TABLE KafkaSource (
  `user_id` VARCHAR,
  `user_name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'kafka',
  'topic' = 'test_source',
  'properties.bootstrap.servers' = 'Kafka的Broker实例业务IP:Kafka端口号',
  'properties.group.id' = 'testGroup',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv',
  'subtask.scan.records-per-second.limit' = '1000'
);
CREATE TABLE KafkaSink(
  `user_id` VARCHAR,
  `user_name` VARCHAR,
  `age` INT
) WITH (
  'connector' = 'kafka',
  'topic' = 'test_sink',
  'properties.bootstrap.servers' = 'Kafka的Broker实例业务IP:Kafka端口号',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'csv'
);
Insert into
  KafkaSink
select
  *
from
  KafkaSource;
```

8.9 FlinkSQL 支持 Multi Join

使用场景

在FlinkSQL多表Join场景中，默认情况下多个Join操作会逐个执行，每个Join操作都会产生独立的中间状态（state），当关联表较多时，中间状态会大量累积，占用大量存储资源并降低作业性能。FlinkSQL支持Multi Join，在多表Join场景下可以将多个Join操作合并为一个Multi Join算子，有效地减少中间状态。

约束与限制

- Multi Join不支持Join-To-Live特性。
- 使用Multi Join前后的快照互不兼容。
- 使用Multi Join多表关联时，表关联键必须一致。

使用方法

可通过以下两种方式配置table.optimizer.multi-join.enabled为true，以开启Multi Join功能：

- **方式一**：配置Flink作业时，在作业flink_config参数中添加该自定义参数。
- **方式二**：在端点的Flink参数中添加该自定义参数。

SQL 示例

```
CREATE TABLE datagen1 (id int, f1 int, PRIMARY KEY (id) NOT ENFORCED)
WITH
```

 版权所有 © 华为云计算技术有限公司

```
  (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );
CREATE TABLE datagen2 (id int, f2 int, PRIMARY KEY (id) NOT ENFORCED)
WITH
  (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );
CREATE TABLE datagen3 (id int, f3 int, PRIMARY KEY (id) NOT ENFORCED)
WITH
  (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );

CREATE TABLE datagen4 (id int, f4 int, PRIMARY KEY (id) NOT ENFORCED)
WITH
  (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );
CREATE TABLE datagen5 (id int, f5 int, PRIMARY KEY (id) NOT ENFORCED)
WITH
  (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );
CREATE TABLE print (id int, f1 int, f2 int, f3 int, f4 int, f5 int)
WITH
  ('connector' = 'print');
insert into
  print
select
  datagen1.id,
  datagen1.f1,
  datagen2.f2,
  datagen3.f3,
  datagen4.f4,
  datagen5.f5

from
  datagen1
  left join datagen2 on datagen1.id = datagen2.id
  and datagen2.id > 2
  join datagen3 on datagen1.id = datagen3.id
  and datagen3.id > 3
  right join datagen5 on datagen1.id = datagen5.id
  and datagen5.id > 5
  full outer join datagen4 on datagen1.id = datagen4.id
  and datagen4.id > 4
  and datagen1.id > 1;
```

Multi Join 算子支持 Empty Result Pruning

使用场景

在Multi Join运行时，支持空结果剪枝。当某个表对应的状态为空时，直接跳过该状态的遍历，从而减少Multi Join中无效的递归扫描，提高查询性能和资源利用率。

使用方法

配置“table.optimizer.multi-join.empty-result-pruning.enabled”的参数值为“true”开启Empty Result Pruning功能。

版权所有 © 华为云计算技术有限公司

8.10 Flink On Iceberg 支持维表 Join

使用场景

Flink On Iceberg支持维表Join，可以将实时流数据与存储在Iceberg中的静态或缓慢变化的维度数据进行关联，从而补齐流记录的数据。

约束与限制

- Lookup Join默认在每个并发加载全量数据，仅适用于小数据量的维表。
- Lookup Join Key不支持ARRAY、BINARY、DECIMAL、TIME、MAP、VARIANT等复杂数据类型。
- Cache refresh不支持增量读取，每次刷新全表扫描。
- Iceberg维表仅支持INSERT、UPDATE_AFTER消息。

参数

Key	默认值	取值范围	含义
lookup.refresh.async	false	true / false	是否异步刷新缓存。 ● true：后台线程刷新，不阻塞查询； ● false：同步刷新阻塞查询。
lookup.refresh.interval	1h	Duration（如 24h）	缓存全量刷新的时间间隔
lookup.cache.type	AUTO	AUTO / HEAP / ROCKSDB	缓存后端类型。 ● AUTO：根据表大小自动选择。 ● HEAP：基于Caffeine堆内存。 ● ROCKSDB：基于磁盘。
lookup.cache.auto.bytes-threshold	64mb	Flink MemorySize（如 32mb）	AUTO模式下，表大小超过此阈值用ROCKSDB，否则用HEAP。
lookup.cache.max-table-size	512mb	Flink MemorySize（如 256mb）	Iceberg维表大小超过此阈值后则无法被加载。
lookup.rocksdb.path	${java.io.tmpdir}/iceberg-lookup-rocksdb	有效本地目录路径。	ROCKSDB模式下数据存储路径。

版权所有 © 华为云计算技术有限公司

使用方法

步骤1 新建一个Iceberg表，例如iceberg_catalog.`default`.iceberg_table的代码示例如下。

```
CREATE TABLE IF NOT EXISTS iceberg_catalog.`default`.iceberg_table (
  id BIGINT,
  data STRING
) PARTITIONED BY (id)
WITH
  ('format-version' = '2');
```

步骤2 维表Join，SQL示例：

```
CREATE TABLE datagen (id BIGINT)
WITH
  (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );

CREATE TABLE print
WITH
  ('connector' = 'print') AS
SELECT
  *
FROM
  datagen AS o
  LEFT JOIN iceberg_catalog.`default`.iceberg_table /*+ OPTIONS('lookup.refresh.interval' = '24h') */ FOR
SYSTEM_TIME AS OF PROCTIME () AS d ON o.id = d.id;
```

----结束

8.11 Flink On Iceberg 支持 VARIANT 数据类型

使用场景

Flink写Iceberg，支持动态变化和嵌套的半结构化数据。

约束与限制

- 仅支持Iceberg V3表。
- 仅支持Iceberg Parquet format。

相关函数

```
VARIANT PARSE_JSON(VARCHAR jsonStr [, BOOLEAN allowDuplicateKeys])
VARIANT TRY_PARSE_JSON(VARCHAR jsonStr [, BOOLEAN allowDuplicateKeys])
```

参数	数据类型	说明
jsonStr	VARCHAR	需要解析的JSON字符串。 如果jsonStr参数值为NULL时，则返回值为NULL。
allowDuplicateKeys	BOOLEAN	可选，取值如下： • true：保留JSON字符串最后一个重复key的值。 • false（默认值）：JSON字符串中如出现重复的key则返回NULL。

版权所有 © 华为云计算技术有限公司

函数	默认错误处理机制	非法JSON输入返回值	容错需求场景
PARSE_JSON	显式报错。	默认抛出异常。	需严格校验输入合法性。
TRY_PARSE_JSON	静默容错。	默认返回NULL。	需兼容异常数据流。

使用方法

步骤1 新建一个Iceberg表，例如iceberg_catalog.`default`.iceberg_table的代码示例如下。

```
CREATE TABLE IF NOT EXISTS iceberg_catalog.`default`.iceberg_table (
  id BIGINT,
  test_data1 VARIANT,
  test_data2 VARIANT
) WITH
  ('format-version' = '3');
```

步骤2 写VARIANT类型，SQL示例：

```
CREATE TABLE datagen (id BIGINT)
WITH
  (
    'connector' = 'datagen',
    'rows-per-second' = '1'
  );

INSERT INTO
  iceberg_catalog.`default`.iceberg_table
SELECT
  id,
  PARSE_JSON (
    '{"user":{"name":"Bob","address":{"city":"Shanghai","zip":"200000"}},"tags":["dev","flink"]}'
  ),
  TRY_PARSE_JSON (
    '{"user":{"name":"Bob","address":{"city":"Shanghai","zip":"200000"}},"tags":["dev","flink"]}'
  )
FROM
  datagen;
```

步骤3 读VARIANT类型，SQL示例：

```
CREATE TABLE print
WITH
  ('connector' = 'print') AS
SELECT
  id,
  JSON_VALUE (test_data1, '$.user.name'),
  JSON_VALUE (test_data2, '$.tags[0]')
FROM
  iceberg_catalog.`default`.iceberg_table /*+ OPTIONS('streaming'='true', 'monitor-interval'='60s') */;
```

----结束

 版权所有 © 华为云计算技术有限公司