ALIBABA CLOUD

Alibaba Cloud

Cloud Native Distributed Database PolarDB-X

SQL Reference

Document Version: 20220601

Alibaba Cloud

Legal disclaimer

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

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

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

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

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

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

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

Document conventions

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

Table of Contents

1.SQL limits

PolarDB-X 1.0is highly compatible with the MySQL protocol and the Structured Query Language (SQL) syntax of MySQL. However, some limits are imposed on the SQL statements for PolarDB-X 1.0. This is because the architecture of distributed databases differs from that of single-instance databases. This topic describes the limits of SQL statements in PolarDB-X 1.0.

General limits on the SQL statements

- PolarDB-X 1.0 does not support custom data types or custom functions.
- PolarDB-X 1.0 does not support stored procedures, triggers, or cursors.
- PolarDB-X 1.0 does not support temporary tables.
- PolarDB-X 1.0 does not support compound statements, such as BEGIN...END, LOOP...END LOOP, REPEAT...UNTIL...END REPEAT, and WHILE...DO...END WHILE.
- PolarDB-X 1.0 does not support flow control statements, such as IF and WHILE statements.
- PolarDB-X 1.0 does not support foreign key.

Limits on the SQL syntax

- DDL
  - You cannot execute the `CREATE TABLE tbl_name LIKE old_tbl_name` statement for table sharding.
  - You cannot execute the `CREATE TABLE tbl_name SELECT statements` for table sharding.
  - You cannot execute the RENAME statement to rename multiple tables at a time.
  - You cannot execute the ALTER TABLE statement to change shard key fields.
  - PolarDB-X 1.0 does not support data definition language (DDL) operations across schemas, such as `CREATE TABLE db_name.tbl_name (...)` .

  For more information about DDL statements, see DDL.
- DML
  - PolarDB-X 1.0 does not support the following statements: SELECT INTO OUTFILE, INTO DUMPFILE, and INTO var_name.
  - PolarDB-X 1.0 does not support STRAIGHT_JOIN or NATURAL JOIN operations.
  - PolarDB-X 1.0 does not support subqueries in UPDATE SET clauses.
  - PolarDB-X 1.0 does not support INSERT DELAYED statements.
  - PolarDB-X 1.0 does not support variable references and operations in SQL statements. For example, you cannot execute the following statement: `SET @c=1, @d=@c+1; SELECT @c, @d` .
  - You cannot perform the INSERT, REPLACE, UPDATE, or DELETE operations on broadcast tables in flexible transactions.

  For more information about data manipulation language (DML) statements, see DML.
- Subqueries
  - PolarDB-X 1.0 does not support subqueries in HAVING or JOIN ON clauses.
  - PolarDB-X 1.0 does not support the ROW functions in the scalar subqueries that use equal signs (=) as operators.

For more information about subqueries, see Subqueries.

- Database management
  - PolarDB-X 1.0 does not support the combination of LIMIT and COUNT in SHOW WARNINGS statements.
  - PolarDB-X 1.0 does not support the combination of LIMIT and COUNT in SHOW ERRORS statements.
- Operators that are not supported by PolarDB-X 1.0

  PolarDB-X 1.0 does not support the assignment operators `':='`

  For more information about operators, see Operators.
- Functions that are not supported by PolarDB-X 1.0
  - Full-text search functions. For more information, see Full-Text Search Functions.
  - XML functions. For more information, see XML Functions.
  - Global transaction identifier (GTID) functions. For more information, see Functions Used with Global Transaction Identifiers (GTIDs)
  - Enterprise encryption functions. For more information, see MySQL Enterprise Encryption.

  For more information about functions, see Functions.
- Keywords that are not supported by PolarDB-X 1.0
  - MILLISECOND
  - MICROSECOND

2.Instructions for sharding function

2.1. Overview

PolarDB-X 1.0 is a database service that supports both database sharding and table sharding. This topic describes the sharding functions of PolarDB-X 1.0.

Sharding method

In PolarDB-X 1.0, the sharding method of a logical table is defined by a sharding function and a sharding key (including the MySQL data type of the key). The sharding function contains the number of shards and the routing algorithm. The database shard and table shard of a logical table in PolarDB-X 1.0 are generated with the same sharding method only when the same sharding function and sharding key are used. If the database shard and table shard are generated with the same sharding method, PolarDB-X 1.0 can locate a unique physical database shard and physical table shard based on the value of the sharding key. If the sharding methods used for the database sharding and table sharding of a logical table are different, and no conditions are specified for database sharding and table sharding in the SQL statement, PolarDB-X 1.0 scans all database shards or all table shards to query data.

Support for database sharding and table sharding

Sharding function	Description	Support database sharding	Support table sharding
HASH	Performs a simple modulo operation.	Yes	Yes
STR_HASH	Returns a substring.	Yes	Yes
UNI_HASH	Performs a simple modulo operation.	Yes	Yes
RIGHT_SHIFT	Performs a signed right shift on the value of the database shard key.	Yes	Yes
RANGE_HASH	Performs hashing when two sharding keys are required.	Yes	Yes
MM	Performs hashing by month.	No	Yes
DD	Performs hashing by date.	No	Yes
WEEK	Performs hashing by week.	No	Yes
MMDD	Performs hashing by month and date.	No	Yes

Sharding function	Description	Support database sharding	Support table sharding
YYYYMM	Performs hashing by year and month.	Yes	Yes
YYYYWEEK	Performs hashing by year and week.	Yes	Yes
YYYYDD	Performs hashing by year and date.	Yes	Yes

Support for global secondary indexes

- PolarDB-X 1.0 supports Global secondary indexes. In terms of data storage, each GSI corresponds to a logical table that stores index data. This table is called an index table.
- PolarDB-X 1.0 also allows you to specify the sharding method of the index table when you create a GSI. Index tables and normal logical tables support the same sharding functions. For more information, see Use global secondary indexes.

Supported data types

Sharding function	Data type										
	INT	BIGINT	MEDIUMINT	SMALLINT	TINYINT	VARCHAR	CHAR	DATE	DATETIME	TIMESTAMP	Other type
HASH	√	√	√	√	√	√	√	×	×	×	×
UNI_HASH	√	√	√	√	√	√	√	×	×	×	×
RANGE_HASH	√	√	√	√	√	√	√	×	×	×	×
RIGHT_SHIFT	√	√	√	√	√	×	×	×	×	×	×
STR_HASH	×	×	×	×	×	√	√	×	×	×	×
MM	×	×	×	×	×	×	×	√	√	√	×
DD	×	×	×	×	×	×	×	√	√	√	×
WEEK	×	×	×	×	×	×	×	√	√	√	×
MMDD	×	×	×	×	×	×	×	√	√	√	×

Sharding function	Data type										
	INT	BIGINT	MEDIUMINT	SMALLINT	TINYINT	VARCHAR	CHAR	DATE	DATETIME	TIMESTAMP	Other type
YYYYMM	×	×	×	×	×	×	×	√	√	√	×
YYYYWEEK	×	×	×	×	×	×	×	√	√	√	×
YYYYDD	×	×	×	×	×	×	×	√	√	√	×

Sharding function syntax

PolarDB-X 1.0 is compatible with the Data Definition Language (DDL) table statements in MySQL. It also provides the `drds_partition_options` keyword for database sharding and table sharding, as shown in the following statements.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [drds_partition_options]
    [partition_options]
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [drds_partition_options]
    [partition_options]
    select_statement
drds_partition_options:
    DBPARTITION BY
        { {HASH|YYYYMM|YYYYWEEK|YYYYDD|...}([column])}
        [TBPARTITION BY
            { {HASH|MM|DD|WEEK|MMDD|YYYYMM|YYYYWEEK|YYYYDD|...}(column)}
          [TBPARTITIONS num]
        ]
```

2.2. HASH

This topic describes how to use the HASH function.

Note

The UNI_HASH functions perform simple modulo operations. The output of the UNI_HASH functions can be evenly distributed only when the values in the partitioning key column are evenly distributed.

Limits

The partitioning key must be an integer or a string.

Routing method

- If different partitioning keys are used to execute the HASH function for database shards and table shards, divide the value of the database shard key by the number of database shards and find the remainder. If the key value is a string, the string is first converted into a hash value and then used for route calculation. For example, `HASH(8)` is equivalent to `8 % D`, where D indicates the number of database shards. `HASH("ABC")` is equivalent to `hashcode("ABC").abs() % D`.
- If the same partitioning key is used to execute the HASH function for database shards and table shards, divide the value of the partitioning key by the total number of table shards and find the remainder. Assume that two database shards are created, each database shard contains four table shards, table shards 0 to 3 are stored in database shard 0, and table shards 4 to 7 are stored in database shard 1. Based on this routing method, the key value 15 is distributed to table shard 7 in database shard 1. The equation is ((15 % (2*4) = 7)).

Scenarios

The HASH function can be used in the following scenarios:

- Partition databases by user ID or order ID.
- Use a string as the partitioning key.

Examples

Assume that you want to execute the HASH function to create non-partitioned tables in database shards based on the ID column. You can execute the following Data Definition Language (DDL) statement to create tables:

```
create table test_hash_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by HASH(ID);
```

2.3. STR_HASH

This topic describes how to use the STR_HASH function.

Note

Table shards that are created with the STR_HASH function are applicable only to point query scenarios. Range queries for a service trigger a full table scan, which causes a slow response.

Limits

- The partitioning key must be a string (CHAR or VARCHAR).
- The parameters of the STR_HASH function cannot be modified after a table is created.
- The version of the PolarDB-X 1.0 instance must be 5.3.5 or later.

Scenarios

- The STR_HASH function is applicable to scenarios that require precise routing where only one table or database shard corresponds to the value of one partitioning key. This value must be a string.

For example, an Internet finance application partitions data into database shards by year and month of the year by using the YYYYMM function, and then partitions data into table shards by order ID. In this application, the last three characters of each order ID are an integer that ranges from 000 to 999. This application is required to route the last three characters of each order ID in a physical database shard to one physical table shard. Therefore, the application uses the YYYYMM function to partition data into database shards and then uses the STR_HASH function to partition data into table shards. To meet the requirements of the application, each database shard contains 1,024 table shards. The following SQL statement is used to create the required shards:

```
create table test_str_hash_tb (
    id int NOT NULL AUTO_INCREMENT,
    order_id varchar(30) NOT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by YYYYMM(`create_time`)
tbpartition by STR_HASH(`order_id`, -1, 3, 1) tbpartitions 1024;
```

This SQL statement finds the last three characters of each order ID, converts them into an integer (000 to 999), and then performs the modulo operation to calculate the appropriate table shard. The total number of table shards is 1,024. The routing result ensures that each physical table shard corresponds to the value of only one partitioning key. The default partitioning function of PolarDB-X 1.0 cannot achieve this effect because the hashCode function may convert strings into integers that are not unique. One physical table shard may correspond to the values of multiple partitioning keys.

- Typical scenarios of point query

  The STR_HASH function is applicable to scenarios where a string is used as the partitioning key. Point query is used in most scenarios, such as querying transaction orders and logistics orders by ID.

Syntax

The STR_HASH function allows you to truncate the string value of a partitioning key into a substring by specifying the start position subscript and end position subscript. Then, the function uses this substring as a string or an integer input to calculate the routes of specific physical database shards and table shards. For more information, see the following syntax:

```
STR_HASH( shardKey [, startIndex, endIndex [, valType [, randSeed ] ] ] )
```

Parameters

Parameter	Description
shardKey	The name of the partitioning key column.
startIndex	The start position subscript of the substring. The positions of the characters in the original string start at 0. This means that the value 0 indicates the first character in the original string. To disable truncation, retain the default value of -1.

Parameter	Description
endIndex	The end position subscript of the target substring. The positions of the characters in the original string start at 0. This means that the value 0 indicates the first character in the original string. To disable truncation, retain the default value of -1. **Note** Note the following values of startIndex and endIndex: • For `startIndex == j && endIndex = k (j>=0, k>=0,k>j)`, the `[ j, k )` range of the original string is used as the substring. For example: ◦ For the ABCDEFG string, the value of the `[1,5)` range is `BCDE`. ◦ For the ABCDEFG string, the value of the `[2,2)` range is `''`. ◦ For the ABCDEFG string, the value of the `[4,100)` range is `EFG`. ◦ For the ABCDEFG string, the value of the `[100,105)` range is `''`. • For `startIndex == -1 && endIndex = k (k>=0)`, the last K characters of the original string are used as the substring. If the original string contains fewer than K characters, the entire string is used as the substring. • For `startIndex = k && endIndex == -1 (k>=0)`, the first K characters of the original string are used as the substring. If the original string contains fewer than K characters, the entire string is used as the substring. • For `startIndex == -1 && endIndex == -1`, no truncation is performed. The substring is the same as the original string.
valType	The type of the substring that is used for calculating routes of database shards and table shards. Valid values: • 0 (default): PolarDB-X 1.0 uses the substring as a string to calculate routes. • 1: PolarDB-X 1.0 uses the substring as an integer to calculate routes. The integer value of the substring cannot be greater than 9223372036854775807 and cannot be a floating-point number.

Parameter	Description
randSeed	The value of a random seed that PolarDB-X 1.0 uses to calculate the hash value of routes when the substring is used as a string. This value is used only when the STR_HASH function cannot achieve even data distribution by using the default random seed. The default value is 31. You can set this parameter to other values such as 131, 13131, and 1313131. **Note** • This parameter can be set only when valType is set to 0. • After you set this parameter, you must redistribute data by manually exporting all data and then importing it with the new partitioning algorithm.

Examples

Assume that the data type of order_id is VARCHAR(32). You want to use order_id as the partitioning key to partition data into four database shards and eight table shards.

- Assume that you want to use the last four characters of the order_id string as an integer to calculate the routes of the database shards and table shards. You can use the following SQL statement to create tables:

```
create table test_str_hash_tb (
    id int NOT NULL AUTO_INCREMENT,
    order_id varchar(32) NOT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by STR_HASH(`order_id`, -1, 4, 1)
tbpartition by STR_HASH(`order_id`, -1, 4, 1) tbpartitions 2;
```

- Assume that you want to use the characters from the third character (startIndex = 2) to the seventh character (endIndex = 7) of the order_id string as a substring to calculate the routes of the database and table shards. You can use the following SQL statement to create tables:

```
create table test_str_hash_tb (
    id int NOT NULL AUTO_INCREMENT,
    order_id varchar(32) NOT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by STR_HASH(`order_id`, 2, 7)
tbpartition by STR_HASH(`order_id`, 2, 7) tbpartitions 2;
```

- Assume that you want to use the first five characters of the order_id string as a substring to calculate the routes of the database shards and table shards. You can use the following SQL statement to create tables:

```
create table test_str_hash_tb (
    id int NOT NULL AUTO_INCREMENT,
    order_id varchar(32) NOT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by STR_HASH(`order_id`, 5, -1)
tbpartition by STR_HASH(`order_id`, 5, -1) tbpartitions 2;
```

FAQ

Q: What are the differences between `dbpartition by STR_HASH(order\_id)` and `dbpartition by HASH(order\_id)`?

A: STR_HASH and HASH both use the value of a string to calculate the hash routes of database shards and table shards. However, they use different route algorithms. STR_HASH allows you to truncate the original string into a substring, which it uses to create tables. STR_HASH also uses the UNI_HASH algorithm when the routes of database shards and table shards are calculated based on the hash value of a string. HASH performs a simple modulo operation to obtain the hash value of a string.

2.4. UNI_HASH

This topic describes how to use UNI_HASH.

Considerations

The UNI_HASH functions perform simple modulo operations. The hash output can be evenly distributed only when the values in the sharding column are evenly distributed.

Limits

- The data type of shard keys must be integers or strings.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1508068 or later.

Routing method

UNI_HASH is used in the following scenarios:

- When you run the UNI_HASH function to perform database sharding, the values of the database shard key are divided by the number of database shards to obtain the remainders. If the key values are strings, the strings are converted to hash values. Then, the hash values are calculated to complete route computing. For example, `HASH('8')` is equivalent to `8 % D`. Letter D indicates the number of database shards.
- Assume that you run the UNI_HASH function to implement database sharding and table sharding by using the same shard key. The values of the database shard key are divided by the number of database shards to obtain the remainders. Then, data is evenly routed to each table shard of the database shard.

Scenarios

- Database sharding is implemented based on user IDs or order IDs.
- The data type of shard keys is an integer or a string.
- The database sharding must be implemented for two logical tables by using the same shard key. The

number of table shards for one table is different from that for the other table. The two tables are often joined by using the shard key.

Use cases

Assume that you must run the UNI_HASH function to implement sharding by using the values of the ID column and each database shard contains four tables. You can execute the following data definition language (DDL) statement to create tables:

```
create table test_hash_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by UNI_HASH(ID)
tbpartition by UNI_HASH(ID) tbpartitions 4;
```

Comparison with HASH

Comparison scenario	UNI_HASH	HASH
Database sharding is implemented and table sharding is not implemented.	The two functions use the same routing method. In this method, the values of the database shard key are divided by the number of database shards to obtain the remainders.	
Sharding is implemented by using the same shard key.	The results that are routed to database shards by using the same key value do not change as the number of table shards changes.	The results that are routed to database shards by using the same key value change as the number of table shards changes.
Sharding must be implemented for two logical tables by using the same shard key. However, the number of table shards for one logical table is different from that for the other logical table.	When one table is joined with the other table by using the shard key, a cross-database join does not occur.	When one table is joined with the other table by using the shard key, a cross-database join occurs.

2.5. RANGE_HASH

This topic describes how to use the RANGE_HASH function.

Scenarios

The RANGE_HASH function is applicable to scenarios where two partitioning keys are required but only one partitioning key value is available during queries.

Limits

- The two partitioning keys must be of the same data type, which can be the string or number type.
- The two partitioning keys cannot be modified.
- The partitioning keys do not support range query.

- When data is inserted, the last N characters of the two partitioning keys must be the same.
- A string must contain no less than N characters.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later.

Routing method

Calculate the hash value based on the last N characters of either partitioning key, then divide by the number of database shards and find the remainder. The letter N indicates the third parameter in the function. For example, during calculation of the `RANGE_HASH(COL1, COL2, N)` function, COL1 is preferentially selected, and then its last N characters are truncated for calculation. If COL1 does not exist, COL2 is selected and truncated for calculation.

Examples

Assume that PolarDB-X 1.0 has eight physical database shards. You want to partition data into database shards by buyer ID and order ID. However, only one of these IDs is available during queries. You can use the following Data Definition Language (DDL) statement to create order tables:

```
create table test_order_tb (
    id int,
    buyer_id varchar(30) DEFAULT NULL,
    order_id varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8
    dbpartition by RANGE_HASH(buyer_id,order_id, 10)
    tbpartition by RANGE_HASH (buyer_id,order_id, 10) tbpartitions 3;
```

2.6. RIGHT_SHIFT

This topic describes how to use the RIGHT_SHIFT function.

Limits

- The shard key must be an integer.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later..

Routing method

The RIGHT_SHIFT function performs a signed right shift on the value of the database shard key. The function then divides the resulting integer by the number of database or table shards and finds the remainder. Note that the value of the shard key must be an integer. You can specify the number of bits to shift by running a data definition language (DDL) statement.

> **Note** The number of bits to shift cannot exceed the number of bits used to represent an integer.

Scenarios

The RIGHT_SHIFT function can produce more even hashing when the lower-digit parts of most shard key values are similar but the higher-digit parts vary greatly.

For example, assume you have the following four shard key values: `0x0100`, `0x0200`, `0x 0300` and `0x0400`. The rightmost eight bits of each value are 0. Services may use the rightmost bits as flags. In this case, using the remainder method on the original values can result in less effective hashing. You can use `RIGHT_SHIFT (shardKey, 8)` to shift the values of the keys eight bits to the right and obtain the following values: `0x01`, `0x02`, `0x03` and `0x04`. These new values result in relatively even hashing. If a database is divided into four shards, each value corresponds to one shard.

Use cases

For example, assume that you are using the ID column as the shard key. You may want to shift the values of this column four bits to the right for hashing purposes. In this case, you can run the following statement:

```
create table test_hash_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by RIGHT_SHIFT(id, 8)
tbpartition by RIGHT_SHIFT(id, 8) tbpartitions 4;
```

2.7. MM

This topic describes how to use the MM function.

Limits

- The partitioning key must be of the DATE, DATETIME, or TIMESTAMP type.
- This function can be used only for partitioning data into table shards, not into database shards.
- When you use the MM function to partition data into table shards, ensure that each database shard has no more than 12 table shards. This is because a year has 12 months.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later..

Routing method

To obtain the table shard subscript, divide by the month of the year in the time value of the database shard key.

Scenarios

The MM function is applicable to scenarios where data needs to be partitioned into table shards by month. The name of a table shard indicates a specific month.

Examples

Assume that you want to partition data into database shards by user ID and then create a physical table shard for each month based on the `create_time` column. You can use the following Data Definition Language (DDL) statement to create tables:

```
create table test_mm_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by HASH(id)
tbpartition by MM(create_time) tbpartitions 12;
```

2.8. DD

This topic describes how to use the DD function.

Limits

- The partitioning key must be of the DATE, DATETIME, or TIMESTAMP type.
- This function can be used only for partitioning data into table shards, not into database shards.
- When you use the DD function to partition data into table shards, ensure that each database shard has no more than 31 table shards. This is because a month cannot have more than 31 days.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later..

Routing method

To obtain the table shard subscript, divide by the day in the time value of the database shard key.

Scenarios

The DD function is applicable to scenarios where data needs to be partitioned into table shards by day. The name of a table shard indicates a specific day.

Examples

Assume that you want to partition data into database shards by user ID and then create a physical table shard for each day based on the `create_time` column. You can use the following Data Definition Language (DDL) statement to create tables:

```
create table test_dd_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by HASH(id)
tbpartition by DD(create_time) tbpartitions 31;
```

2.9. WEEK

This topic describes how to use the WEEK function.

Limits

- The partitioning key must be of the DATE, DATETIME, or TIMESTAMP type.
- This function can be used only for partitioning data into table shards, not into database shards.
- When you use the WEEK function to partition data into table shards, ensure that each database shard has no more than seven table shards. This is because a week has seven days.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later..

Routing method

To obtain the table shard subscript, divide by the day of the week in the time value of the database shard key.

Scenarios

The WEEK function is applicable to scenarios where data needs to be partitioned into table shards by each day of a week. The name subscript of a table shard indicates a specific day of a week, from Monday to Sunday.

Examples

Assume that you want to partition data into database shards by user ID and then create a physical table shard for each day of the week (Monday to Sunday) based on the `create_time` column. You can use the following Data Definition Language (DDL) statement to create tables:

```
create table test_week_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by HASH(name)
tbpartition by WEEK(create_time) tbpartitions 7;
```

2.10. MMDD

This topic describes how to use the MMDD function.

Limits

- The partitioning key must be of the DATE, DATETIME, or TIMESTAMP type.
- This function can be used only for partitioning data into table shards, not into database shards.
- When you use the MMDD function to partition data into table shards, ensure that each database shard has no more than 366 table shards. This is because a year cannot have more than 366 days.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later..

Routing method

To obtain the table shard subscript, divide by the day in the time value of the partitioning key of a database shard.

Scenarios

The MMDD function is applicable to scenarios where data needs to be partitioned into table shards by day of a year. The name subscript of a table shard indicates a specific day of a year.

Examples

Assume that you want to partition data into database shards by user ID and then create a physical table shard for each day and month based on the `create_time` column. You can use the following Data Definition Language (DDL) statement to create tables:

```
create table test_mmdd_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by HASH(name)
tbpartition by MMDD(create_time) tbpartitions 366;
```

2.11. YYYYDD

This topic describes how to use the YYYYDD function.

Limits

- A shard key must be of the DATE, DATETIME, or TIMESTAMP type.
- Before you use the YYYYDD function, determine the total number of physical table shards that are required based on a specific cycle, such as two years. The YYYYDD function can be used to create only one table shard for each day within a cycle.
- When a date recurs in the next cycle, data generated on the date may be routed to the same table shards that store data generated on the same date in the last cycle. For example, if you specify a two-year cycle starting from March 1, 2012, data generated on March 1, 2014 in the next cycle may be routed to the same table shard that stores data generated on March 1, 2012. The table shard to which the data is routed depends on the number of table shards.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later.

Routing method

You can use the YYYYDD function to calculate hash values based on the years and days in the time values of a database shard key. Then, the function divides the hash values by the number of database shards to obtain the remainders. As a result, data is partitioned based on the remainders.

For example, if you specify parameters for the YYYYDD function in the `YYYYDD('2012-12-31 12:12:12')` format, the remainder is calculated based on the following formula: `(2012 x 366 + 366)%D`, in which D indicates the number of database shards. The calculation result indicates that December 31, 2012 is the 366th day of year 2012.

Scenarios

The YYYYDD function is suitable for scenarios in which data needs to be partitioned into database shards by year and day of the year. We recommend that you use this function with `tbpartition by YYYYDD(ShardKey)`.

Example

In this example, the PolarDB-X 1.0 instance has two nodes. By default, each node has eight database shards. The data must be partitioned based on the following requirements:

- Data is partitioned into the database shards by year and day of the year.
- Data generated on the same day is partitioned into the same table shard. Each day within two years corresponds to an independent table shard.
- A query is directly distributed to a specific physical table shard of a database shard if shard keys are specified in the query.

The YYYYDD function can meet the preceding requirements. You require that each day within two years correspond to a table shard. Therefore, a total of 732 (366 x 2) physical table shards must be created because a year has up to 366 days. A PolarDB-X 1.0 instance has 16 database shards. The number of physical table shards in each database shard is calculated in the following two steps: 1. Divide the total number of physical table shards that must be created by the number of database shards. 2. Round the result up to the next nearest integer. In this case, the number of physic table shards is 46, which is the nearest integer to 45.75 (732/16). We recommend that the number of table shards be an integer multiple of the number of database shards.

You can use the following Data Definition Language (DDL) statement to create tables:

```
create table test_yyyydd_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by YYYYDD(create_time)
tbpartition by YYYYDD(create_time) tbpartitions 46;
```

2.12. YYYYMM

This topic describes how to use the YYYYMM function.

Limits

- The partitioning key must be of the DATE, DATETIME, or TIMESTAMP type.
- Before you use the YYYYMM function, you must determine the total number of physical table shards required. This number can be determined based on a cycle, such as 2 years. The YYYYMM function can be used only to create a separate table shard for each month within a cycle.
- When a month reoccurs after a cycle has completed, data from that month may be routed to the same table shard in the same database shard. For example, with a two-year cycle starting in March 2012, data from March 2014 may be routed to the same table shard as data from March 2012. The specific table shard to which the data is routed depends on the number of table shards.
- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later..

Routing method

Calculate the hash value based on the year and month of the time value of the database shard key. Then divide by the number of database shards and find the remainder.

For example, the `YYYYMM('2012-12-31 12:12:12')` function is equivalent to `(2012 × 12 + 12) % D`, where D indicates the number of database shards. The calculation result indicates that December 31, 2012 is in the 12nd month of 2012.

Scenarios

The YYYYMM function is applicable to scenarios where data needs to be partitioned into database shards by year and month of the year. We recommend that you use this function with `tbpartition YYYYMM(ShardKey)`.

Examples

Assume that a PolarDB-X 1.0 instance has eight physical database shards and that you have the following requirements:

- Partition data into the database shards by year and month of the year.
- Distribute data from the same month to the same table shard and ensure that each month within two years corresponds to a separate table shard.
- Directly distribute a query by partitioning key of database shards and table shards to a specific physical table shard of a database shard.

The YYYYMM function can meet the preceding requirements. You require that each month within two years correspond to a table shard. Therefore, a total of 24 physical table shards must be created, because a year has 12 months (12 × 2 = 24).PolarDB-X 1.0 has eight database shards. Therefore, three physical table shards must be created in each database shard (24/8 = 3).

You can use the following Data Definition Language (DDL) statement to create tables:

```
create table test_yyyymm_tb (
        id int,
        name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by YYYYMM(create_time)
tbpartition by YYYYMM(create_time) tbpartitions 3;
```

2.13. YYYYWEEK

This topic describes how to use the YYYYWEEK function.

Limits

- The partitioning key must be of the DATE, DATETIME, or TIMESTAMP type.
- Before you use the YYYYWEEK function,you must determine the total number of physical table shards required. This number can be determined based on a cycle, such as 2 years. The YYYYWEEK function can be used only to create a separate table shard for each week within a cycle.
- When a week reoccurs after a cycle has completed, data from that week may be routed to the same table shard in the same database shard. For example, with a two-year cycle starting on the first week of 2012, data from the first week of 2014 may be routed to the same table shard as data from the first week of 2012. The specific table shard to which the data is routed depends on the number of table shards.

- The version of the PolarDB-X 1.0 instance must be 5.1.28-1320920 or later..

Routing method

Calculate the hash value based on the year and week of the time value of the database shard key. Then divide by the number of database shards and find the remainder.

For example, the `YYYYWEEK('2012-12-31 12:12:12')` function is equivalent to `(2013 × 54 + 1) % D`, where D indicates the number of database shards. The calculation result indicates that the December 31, 2012 is in the first week of 2013.

Scenarios

The YYYYWEEK function is applicable to scenarios where data needs to be partitioned into database shards by year and week of the year. We recommend that you use this function with `tbpartition by YYYYWEEK(ShardKey)`.

Examples

Assume that a PolarDB-X 1.0 instance has eight physical database shards and that you have the following requirements:

- Partition data into the database shards by year and week of the year.
- Distribute data from the same week to the same table shard and ensure that each week within two years corresponds to a separate table shard.
- Directly distribute a query by partitioning key of database shards and table shards to a specific physical table shard of a database shard.

The YYYYWEEK function can meet the preceding requirements. You require that each week within two years correspond to a table shard. Therefore, a total of 106 physical table shards must be created, because a year has up to 53 weeks (53 × 2 = 106). PolarDB-X 1.0 has eight database shards. Therefore, 14 physical table shards must be created in each database shard (106/8 = 13.25, rounded to 14). We recommend that the number of table shards be an integer multiple of the number of database shards.

You can use the following Data Definition Language (DDL) statement to create tables:

```
create table test_yyyyweek_tb (
    id int,
    name varchar(30) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
dbpartition by YYYYWEEK(create_time)
tbpartition by YYYYWEEK(create_time) tbpartitions 14;
```

3.Manage DDL tasks

3.1. Overview

PolarDB-X 1.0 introduces a DDL execution engine that is used by the DRDS instances of V5.3.12 or later to support the DDL task management feature. This feature allows you to view the status of DDL execution tasks, resume the failed DDL tasks, and roll back the failed DDL tasks.

Concepts related to DDL task management

Before you use the DDL task management feature, we recommend that you understand the following concepts:

- DDL task: a task that corresponds to the process of executing a DDL statement.
- Management statement: an SQL statement that is specific to PolarDB-X 1.0 and is executed to view or manage DDL tasks.
- Task ID: the unique identifier of a DDL task. A task ID is a 64-bit long signed integer.
- Task status: the status of a DDL task.

For more information about the syntax of task management statements and how to use these statements, see Job management statements.

3.2. Job management statements

Job management statements are extended Structured Query Language (SQL) statements dedicated to PolarDB-X 1.0. They can be used to query the details of data definition language (DDL) jobs and resume or roll back failed DDL jobs. This topic describes the syntax and usage of job management statements.

Query a job

You can query the details of a DDL job in the DDL queue where the DDL job may be either in a non-PENDING state as it is being executed or in the PENDING state due to failures.

> **Note** Jobs that have been executed are in the COMPLETED state and are automatically cleared. You cannot query these jobs by executing the SHOW DDL statement.

- Syntax

```
SHOW [FULL] DDL
```

Parameter	Description

Parameter	Description
FULL	Queries all information of a DDL job. If you do not specify this parameter, only the following common information is displayed. ○ JOB_ID ○ OBJECT_SCHEMA ○ OBJECT_NAME ○ JOB_TYPE ○ PHASE ○ STATE ○ PROGRESS ○ START_TIME ○ END_TIME ○ ELAPSED_TIME ○ REMARK ○ PHY_PROCESS ○ BACKFILL_PROGRESS

- Description of the fields in the result set

Field	Description
JOB_ID	The unique ID of the DDL job. It is a long 64-bit signed integer.
PARENT_JOB_ID	The unique ID of the DDL parent job. It is a long 64-bit signed integer. **Note** If no parent job exists, this field is set to 0.
SERVER	The information of the DRDS server node that executes the DDL job.
OBJECT_SCHEMA	The schema name of the object corresponding to the DDL job. For example, this field can be the name of the current database.
OBJECT_NAME	The name of the object corresponding to the DDL job. For example, this field can be the name of the table where the current DDL statement is executed.
NEW_OBJECT_NAME	The new name of the object corresponding to the DDL job. **Note** This field is valid only when you execute RENAME TABLE. It indicates the target table name.
JOB_TYPE	The type of the DDL job.
PHASE	The phase where the DDL job is located.
STATE	The status of the DDL job.

Field	Description
PROGRESS	The progress of the DDL job.
START_TIME	The time when the execution of the DDL job started.
END_TIME	The time when the execution of the DDL job ended.
ELAPSED_TIME	The time elapsed after the execution of the DDL job ended. Unit: milliseconds.
DDL_STMT	The original DDL statement.
REMARK	The remarks of the DDL job. **Note** This field displays the failure cause of the DDL job when the DDL job is in the PENDING state.

- Example

  Create a logical table that is partitioned into table shards in a database shard. Query the details of the job when the job is being executed.

  i. Execute the CREATE TABLE DDL statement on a connection.

  ```
  mysql> create table test_mdb_mtb (c1 int not null auto_increment primary key, c2 varc
  har(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64;
  ```

  ii. Query the details of the DDL job on another connection.

  ```
  mysql> show full ddl\G
  *************************** 1. row ***************************
      JOB_ID: 110379207557895782 4
  PARENT_JOB_ID: 0
      SERVER: 1:102:10.81.69.55
  OBJECT_SCHEMA: ddltest
  OBJECT_NAME: test_mdb_mtb
  NEW_OBJECT_NAME:
    JOB_TYPE: CREATE_TABLE
       PHASE: EXECUTE
       STATE: RUNNING
    PROGRESS: 90%
  START_TIME: 2019-08-29 14:29:58.787
    END_TIME: 2019-08-29 14:30:07.177
  ELAPSED_TIME(MS): 8416
    DDL_STMT: create table test_mdb_mtb (c1 int not null auto_increment primary key, c2
  varchar(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64
      REMARK:
  ```

Resume a job

You can resume a pending DDL job that is suspended due to failures.

> **Note** Before you resume the job, execute the SHOW DDL statement to check the causes for the interruption or failure. Resume the job only after the failure causes are eliminated. Otherwise, the same problem persists when you attempt to resume the job.

- Syntax

```
RECOVER DDL { ALL | <job_id> [ , <job_id> ] ... }
```

Parameter	Description
ALL	Resumes all DDL jobs that are in the PENDING state. Note that this parameter causes the pending DDL jobs to be executed serially. Use it with caution.
job_id	The ID of the pending DDL job. This ID is displayed in the execution result of the SHOW DDL statement.

- Example

  Create a logical table that is partitioned into table shards in a database shard and interrupt the job during execution. Execute the SHOW DDL statement to query the status and `job_id` of the job. Then, execute the RECOVER DDL statement to resume the job until the table is created.

  i. Interrupt the CREATE TABLE DDL job during execution.

```
mysql> create table test_mdb_mtb (c1 int not null auto_increment primary key, c2 varc
har(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64;
^C^C -- query aborted
```

  ii. Query the information about the DDL job. The interrupted DDL job is in the PENDING state.

```
mysql> show ddl\G
*************************** 1. row ***************************
     JOB_ID: 1103796219480006656
OBJECT_SCHEMA: ddltest
OBJECT_NAME: test_mdb_mtb
  JOB_TYPE: CREATE_TABLE
      PHASE: EXECUTE
      STATE: PENDING
  PROGRESS: 33%
START_TIME: 2019-08-29 14:46:26.769
  END_TIME: 2019-08-29 14:46:29.691
ELAPSED_TIME(MS): 2922
  DDL_STMT: create table test_mdb_mtb (c1 int not null auto_increment primary key, c2
varchar(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64
     REMARK: The job has been interrupted unexpectedly
```

  iii. Execute the RECOVER DDL statement to resume the job.

```
mysql> recover ddl 1103796219480006656;
Query OK, 0 rows affected (7.28 sec)
```

  iv. Execute CHECK TABLE to check the consistency of the table.

```
mysql> check table test_mdb_mtb;
+---------------------------------------+-------+----------+----------+
| TABLE                                 | OP    | MSG_TYPE | MSG_TEXT |
+---------------------------------------+-------+----------+----------+
| ddltest_1562056402230oymk.test_mdb_mtb | check | status   | OK       |
+---------------------------------------+-------+----------+----------+
1 row in set (2.24 sec)
```

Roll back a job

You can roll back a pending DDL job that is suspended due to failures.

> **Note** You can only roll back CREATE TABLE and RENAME TABLE DDL jobs. For other DDL jobs that cannot be rolled back, we recommend that you resume the pending DDL jobs before you perform other DDL operations.

- Syntax

```
ROLLBACK DDL <job_id> [ , <job_id> ] ...
```

Parameter	Description
job_id	The ID of the pending DDL job. This ID is displayed in the execution result of the SHOW DDL statement.

- Example

  Create a logical table that is partitioned into table shards in a database shard and interrupt the job during execution. Execute the SHOW DDL statement to query the status and `job_id` of the job. Then, execute the ROLLBACK DDL statement to roll back the job.

  i. Interrupt the CREATE TABLE DDL job during execution.

  ```
  mysql> create table test_mdb_mtb (c1 int not null auto_increment primary key, c2 varc
  har(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64;
  ^C^C -- query aborted
  ```

  ii. Query the information about the DDL job. The interrupted DDL job is in the PENDING state.

```
mysql> show ddl\G
*************************** 1. row ***************************
     JOB_ID: 1103797850607083520
OBJECT_SCHEMA: ddltest
OBJECT_NAME: test_mdb_mtb
  JOB_TYPE: CREATE_TABLE
      PHASE: EXECUTE
      STATE: PENDING
  PROGRESS: 40%
START_TIME: 2019-08-29 14:52:55.660
  END_TIME: 2019-08-29 14:52:58.885
ELAPSED_TIME(MS): 3225
  DDL_STMT: create table test_mdb_mtb (c1 int not null auto_increment primary key, c2
varchar(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64
     REMARK: The job has been interrupted unexpectedly
```

iii. Execute the ROLLBACK DDL statement to roll back the job.

```
mysql> rollback ddl 1103797850607083520;
Query OK, 0 rows affected (6.42 sec)
```

iv. Rollback is successful. The table does not exist.

```
mysql> show tables like 'test_mdb_mtb';
Empty set (0.00 sec)
```

Cancel a job

You can cancel a running DDL job that is not in the PENDING state.

- Syntax

```
CANCEL DDL <job_id> [ , <job_id> ] ...
```

Parameter	Description
job_id	The ID of the DDL job that is not in the PENDING state. This ID is displayed in the execution result of the SHOW DDL statement.

- Example

  Create a logical table that is partitioned into table shards in a database shard. Execute the CANCEL DDL statement to cancel the job. Execute the SHOW DDL statement to query the status and `job_id` of the job. Later, you can resume or roll back the job.

  i. Execute the CREATE TABLE DDL statement on a connection.

```
mysql> create table test_mdb_mtb (c1 int not null auto_increment primary key, c2 varc
har(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64;
```

  ii. Query the information of the running DDL job by executing the SHOW DDL statement on another connection.

```
mysql> show ddl\G
*************************** 1. row ***************************
   JOB_ID: 1103798959568478208
OBJECT_SCHEMA: ddltest
OBJECT_NAME: test_mdb_mtb
 JOB_TYPE: CREATE_TABLE
     PHASE: EXECUTE
     STATE: RUNNING
 PROGRESS: 26%
START_TIME: 2019-08-29 14:57:20.058
 END_TIME: 2019-08-29 14:57:22.284
ELAPSED_TIME(MS): 2243
 DDL_STMT: create table test_mdb_mtb (c1 int not null auto_increment primary key, c2
varchar(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64
   REMARK:
```

iii. Execute the CANCEL DDL statement to cancel the execution of the DDL job.

```
mysql> cancel ddl 1103798959568478208;
Query OK, 2 rows affected (0.03 sec)
```

iv. Execute the SHOW DDL statement to query the status of the DDL job. The DDL job has been canceled and is in the PENDING state.

```
mysql> show ddl\G
*************************** 1. row ***************************
   JOB_ID: 1103798959568478208
OBJECT_SCHEMA: ddltest
OBJECT_NAME: test_mdb_mtb
 JOB_TYPE: CREATE_TABLE
     PHASE: EXECUTE
     STATE: PENDING
 PROGRESS: 87%
START_TIME: 2019-08-29 14:57:20.058
 END_TIME: 2019-08-29 14:57:28.899
ELAPSED_TIME(MS): 8841
 DDL_STMT: create table test_mdb_mtb (c1 int not null auto_increment primary key, c2
varchar(10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 64
   REMARK: ERR-CODE: [TDDL-4636][ERR_DDL_JOB_ERROR] The job '1103798959568478208' has
been cancelled.
```

Delete a job

You can delete a pending DDL job that is suspended due to failures, and clear the corresponding caches.

> **Warning** Be cautious about executing REMOVE DDL to delete a DDL job. After you delete a pending job, the intermediate state during the DDL execution is exposed, causing disturbance to subsequent operations. Therefore, when you are not sure whether the pending job can be securely deleted, do not execute the REMOVE DDL statement to delete the job. You can preferably resume or roll back the job to make the job exit the PENDING state first.

- Syntax

```
REMOVE DDL { ALL PENDING | <job_id> [ , <job_id> ] ... }
```

Parameter	Description
ALL PENDING	Deletes all jobs that are in the PENDING state and clears internal caches.
job_id	The ID of the pending DDL job. This ID is displayed in the execution result of the SHOW DDL statement.

- Example

  Assume that two tables exist in the database and a referential integrity relationship is established between the two tables. When you attempt to delete the parent table, an error is reported because tables with the referential integrity constraint cannot be deleted. In this case, if you do not want another attempt to delete the table, you can delete the DDL job.

  i. In the database, create two parent-child tables with the referential integrity relationship.

```
mysql> show create table test_parent\G
*************************** 1. row ***************************
Table: test_parent
Create Table: CREATE TABLE `test_parent` (
`id` int(11) NOT NULL,
`pkey` int(11) NOT NULL,
`col` int(11) DEFAULT NULL,
PRIMARY KEY (`id`,`pkey`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`id`)
1 row in set (0.01 sec)
mysql> show create table test_child\G
*************************** 1. row ***************************
Table: test_child
Create Table: CREATE TABLE `test_child` (
`id` int(11) DEFAULT NULL,
`parent_id` int(11) DEFAULT NULL,
KEY `parent_id` (`parent_id`),
CONSTRAINT `test_child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `test_parent` (`i
d`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`parent_id`)
1 row in set (0.02 sec)
```

  ii. Attempt to delete the parent table. Due to the referential integrity constraint, an error is reported.

```
mysql> drop table test_parent;
ERROR 4636 (HY000): [f518265d0066000][10.81.69.55:3306][ddltest]ERR-CODE: [TDDL-4636]
[ERR_DDL_JOB_ERROR] Not all physical operations have been done successfully: expected
9,
but done 0. Caused by: 1217:DDLTEST_1562056402230OYMK_7WW7_0007:Cannot delete or upda
te a parent row: a foreign key constraint fails on `test_parent`;1217:DDLTEST_1562056
4022
30OYMK_7WW7_0000:Cannot delete or update a parent row: a foreign key constraint fails
on `test_parent`;1217:DDLTEST_1562056402230OYMK_7WW7_0002:Cannot delete or update a p
are
nt row: a
```

iii. Query the DDL job.

```
mysql> show ddl\G
*************************** 1. row ***************************
     JOB_ID: 1103806757547171840
OBJECT_SCHEMA: ddltest
OBJECT_NAME: test_parent
  JOB_TYPE: DROP_TABLE
      PHASE: EXECUTE
      STATE: PENDING
  PROGRESS: 0%
START_TIME: 2019-08-29 15:28:19.240
  END_TIME: 2019-08-29 15:28:19.456
ELAPSED_TIME(MS): 216
  DDL_STMT: drop table test_parent
     REMARK: ERR-CODE: [TDDL-4636][ERR_DDL_JOB_ERROR] Not all physical operations hav
e been done successfully: expected 9, but done 0. Caused by: 1217:DDLTEST_1562056402
230OYMK_7WW7_0007:Cannot delete or update a parent row: a foreign key constraint fail
s on `test_pare ...
```

iv. The DDL job violates the referential integrity constraint when you attempt to delete the table. As a result, the delete operation fails. At this time, if you execute CHECK TABLE, you can see that the table is still consistent.

```
mysql> check table test_parent;
+--------------------------------------+-------+----------+----------+
| TABLE                                | OP    | MSG_TYPE | MSG_TEXT |
+--------------------------------------+-------+----------+----------+
| ddltest_1562056402230oymk.test_parent | check | status   | OK       |
+--------------------------------------+-------+----------+----------+
1 row in set (0.05 sec)
```

v. However, the table is inaccessible because a pending job exists for the table.

```
mysql> show tables like 'test_parent';
Empty set (0.00 sec)
mysql> show create table test_parent;
ERROR 4642 (HY000): [f5185a78b066000][10.81.69.55:3306][ddltest]ERR-CODE: [TDDL-4642]
[ERR_UNKNOWN_TABLE] Unknown table 'ddltest.test_parent'
```

vi. In this case, the table deletion job is not executed and the table structure is still consistent. It seems that you can choose to roll back the failed DDL operation. However, the DROP TABLE statement does not allow rollback operations. Therefore, you must choose to delete the failed DDL job.

```
mysql> remove ddl 1103806757547171840;
Query OK, 1 row affected (0.02 sec)
```

vii. After the DDL job is deleted, the table recovers to be accessible.

```
mysql> show tables like 'test_parent';
+-------------------+
| TABLES_IN_DDLTEST |
+-------------------+
| test_parent       |
+-------------------+
1 row in set (0.01 sec)
```

3.3. Control parameters for DDL execution engine

You can change the operations of the data definition language (DDL) execution engine by setting related parameters. This topic describes how to set parameters related to the DDL execution engine.

Parameters

You can set the following parameters in the PolarDB-X 1.0 console as needed.

Parameter	Scope of impact	Default value
ENABLE_ASYNC_DDL	Databases and statements	TRUE (enabled)
PURE_ASYNC_DDL_MODE	Databases, sessions, and statements	FALSE (disabled)
MAX_TABLE_PARTITIONS_PER_DB	Databases and statements	128

ENABLE_ASYNC_DDL

- Description
  - This parameter is set to TRUE by default, indicating that the new DDL execution engine is used.
  - If you set this parameter to FALSE, PolarDB-X 1.0 uses the DDL execution engine of a version earlier than v5.3.12, and the PURE_ASYNC_DDL_MODE and MAX_TABLE_PARTITIONS_PER_DB parameters do not take effect. We recommend that you submit a ticket to determine whether to set this parameter to FALSE. For more information, see Submit a ticket.
- Usage
  - Database level: Set this parameter on the Parameter Settings page in the PolarDB-X 1.0 console. The value that you set takes effect for the entire database. For more information, see Set parameters.
  - Statement level: Add a hint, such as `/*+TDDL:cmd_extra(ENABLE_ASYNC_DDL=FALSE)*/`, at the beginning of a DDL statement, so that this parameter can take effect only for this statement.

PURE_ASYNC_DDL_MODE

- Description
  - This parameter takes effect only when `ENABLE_ASYNC_DDL` is set to TRUE.

- When you set this parameter to FALSE, the client connects to PolarDB-X 1.0 to execute a DDL statement in synchronous blocking mode. In this way, the client returns a response after it completes the DDL job. After the client is disconnected from PolarDB-X 1.0, the ongoing DDL job may be interrupted.
- When you set this parameter to TRUE, the client connects to PolarDB-X 1.0 to execute a DDL statement in asynchronous mode. In this way, the client returns a response when a DDL request is received, and the DDL job continues to be run in the background. You can run the SHOW DDL statement to view the status of the DDL job. For more information about how to use this statement, see Job management statements.
- We recommend that you set this parameter to TRUE when enabling asynchronous mode is explicitly required to prevent unexpected disconnection between the client and PolarDB-X 1.0. Otherwise, we recommend that you set this parameter to its default value (FALSE) to ensure that the DDL operations in PolarDB-X DRDS are compatible with those in the ApsaraDB RDS for MySQL instance.

- Usage
  - Database level: Set this parameter on the Parameter Settings page in the PolarDB-X 1.0 console. The value that you set takes effect for the entire database. For more information, see Set parameters.
  - Session level:
    - After the client connects to PolarDB-X 1.0, execute the `set PURE_ASYNC_DDL_MODE=true` or `set PURE_ASYNC_DDL_MODE=1` statement to enable the asynchronous mode for this session.
    - Execute the `set PURE_ASYNC_DDL_MODE=false` or `set PURE_ASYNC_DDL_MODE=0` statement to restore the default synchronous mode for this session.
  - Statement level: Add a hint, such as `/*+TDDL:cmd_extra(PURE_ASYNC_DDL_MODE=TRUE)*/`, at the beginning of a DDL statement, so that this parameter can take effect only for this statement.

MAX_TABLE_PARTITIONS_PER_DB

- Description
  - This parameter takes effect only when `ENABLE_ASYNC_DDL` is set to TRUE.
  - If the number of table shards in a single physical database exceeds the limit specified by this parameter, the DDL job stops and an error is reported.

    > **Note** The value range of this parameter is 1 to 65535. The default value is 128.

- Usage
  - Database level: Set this parameter on the Parameter Settings page in the PolarDB-X 1.0 console. The value that you set takes effect for the entire database. For more information, see Set parameters.
  - Statement level: Add a hint, such as `/*+TDDL:cmd_extra(MAX_TABLE_PARTITIONS_PER_DB=400)*/`, at the beginning of a DDL statement, so that this parameter can take effect only for this statement.

3.4. Considerations and limits

The new DDL execution engine introduces the task management feature. As a result, statement behavior is different from that in earlier versions. This topic describes the considerations and limits of the task management feature.

Considerations

- If the DDL statement of a DDL task is executed, you can ignore the status of the DDL task. The DDL tasks that are executed are automatically deleted.
- After the DDL statement of a DDL task is executed, we recommend that you immediately execute the CHECK TABLE statement to check the consistency of the logical tables that correspond to the DDL task.
- After you execute a DDL task management statement to resume, roll back, or delete a DDL task, we recommend that you execute the CHECK TABLE statement to check the consistency of the logical tables that correspond to the DDL task.
- If a DDL statement fails to be executed, an error code and an error message are returned. You can also execute the SHOW DDL statement to view the cause of the failure in the pending DDL task. The cause of the failure is indicated by the `REMARK` field.

  > **Notice** We recommend that you identify the cause of the failure in a DDL task and how to handle the failure before you execute a DDL task management statement to resume, roll back, or delete the DDL task. Otherwise, the DDL task management statement may fail to be executed.

- If a DDL statement fails to be executed and the corresponding DDL task is in the pending state, the status of the table that you want to use becomes inaccessible for security considerations. Then, no responses are displayed after a statement such as SHOW TABLES is executed. An error may occur after you execute a DML statement or perform another operation. The error message may indicate that `the table is unknown` or `does not exist`. The table that you want to use can be accessed only after the pending DDL task is resumed or rolled back to make this table enter the consistent state.
- If you use the `IF NOT EXISTS` clause in the CREATE TABLE statement or use the `IF EXISTS` clause in the DROP TABLE statement, some errors that occur during execution do not cause the execution of the DDL statement to fail. However, the errors are recorded in `warnings`. Check whether a message that indicates the number of `warnings` is returned after you execute the DDL statement. For example, the following message may be returned: `1 warning`. Execute the SHOW WARNINGS statement to check the warnings. This way, you can avoid missing important information.
- On a client such as the Data Management (DMS) client,you can configure `PURE_ASYNC_DDL_MODE` to execute DDL statements in asynchronous mode. This prevents the DDL statement execution from being interrupted due to timeout. This configuration is suitable for scenarios in which the time required to execute a DDL statement cannot be estimated and the timeout-based interruption for the connection between the client and the PolarDB-X 1.0 instance is enabled on the client. After the DDL statement is executed, you can execute the `SHOW DDL` statement on the client to view the status of the corresponding DDL task.

Limits

- Only the CREATE TABLE and RENAME TABLE operations can be rolled back.
- The RECOVER DDL and ROLLBACK DDL statements cannot be combined or repeated for a pending DDL task. For example, you cannot execute ROLLBACK DDL to roll back a failed task and then execute RECOVER DDL to recover the task after the rollback fails. Such combinations of operations may cause

inconsistency in the logical tables. If operation combinations are required to meet your business needs, Submit a ticket.

- Execute the `REMOVE DDL` statement only when the database security is ensured. If you execute the `REMOVE DDL` statement when the database security is not ensured, the intermediate states of DDL tasks may be revealed and the logical tables may be inconsistent. If problems occur or the security of the table data is compromised due to the misuse of `REMOVE DDL`, Submit a ticket.
- By default, a maximum of 128 table shards can be created for a single physical database. You can change the limit by using the parameters shown in the following sample code.

```
mysql> create table test_mdb_mtb (c1 int not null auto_increment primary key, c2 varchar(
10), c3 date) dbpartition by hash(c1) tbpartition by hash(c1) tbpartitions 129;
ERROR 4647 (HY000): [f5bd90594800000][30.25.86.55:8527][JICHEN_LOCAL_APP]ERR-CODE: [TDDL-
4647][ERR_TABLE_PARTITIONS_EXCEED_LIMIT] The number of table partitions '129' exceeds the
upper limit '128'. Please specify less table partitions or adjust the value of the parame
ter MAX_TABLE_PARTITIONS_PER_DB.
mysql> /*+TDDL:cmd_extra(MAX_TABLE_PARTITIONS_PER_DB=400)*/create table test_mdb_mtb (c1
int not null auto_increment primary key, c2 varchar(10), c3 date) dbpartition by hash(c1)
tbpartition by hash(c1) tbpartitions 129;
Query OK, 0 rows affected (2.64 sec)
```

- A maximum of 65,535 pending DDL tasks can be accumulated in the task queue of the DDL execution engine. If the number of pending DDL tasks exceeds this limit, you cannot execute DDL statements. In this case, you must execute `REMOVE DDL` to remove unwanted pending tasks. This limit cannot be changed by modifying parameters.

3.5. Best practices

This topic describes the best practices for processing a job in the PENDING state.

Background

In this scenario, an engine is started for a new DDL job. If the DDL job fails or is interrupted due to exceptions, the job enters the PENDING state. In this case, you must take measures to process and resume the job. Otherwise, subsequent DDL statements in this job cannot be executed and an error is returned.

How it works

- You can execute the `SHOW [FULL] DDL` statement to query DDL job details and identify the failure cause. You can view an error message in the `REMARK` field.
- The following list describes the common methods for processing a job in the PENDING state. You can select a method based on your business requirements.
  - Troubleshoot and resolve an issue. For example, check whether the issue is caused by errors in the data. If the issue is caused by duplicates, remove duplicates. If the issue persists, check whether the issue is caused by limits. If the issue is caused by limits, check whether you can lift the limits. After you resolve the issue, execute the `RECOVER DDL` statement to resume the job in the PENDING state.

- If you cannot resolve the issue that causes a job failure, the DDL statements in the job cannot be executed. In this case, you can execute the REMOVE DDL statement to delete the job. Before you execute the REMOVE DDL statement, ensure that the DDL statements in the job are not executed. Otherwise, the table status that is returned by the SHOW statement may be inconsistent with the actual table status. After you delete the DDL job, the table can be queried again.
- If you want to delete a table that a DDL job fails to query, you can execute the `REMOVE DDL` statement to delete the job. Then, you can execute the `DROP TABLE IF EXISTS` statement to delete the table. Before you delete a table, make sure that the table is empty or the data on the table is no longer needed. You must use the `IF EXISTS` keyword in the `DROP TABLE` statement. This ensures that the table can be forcibly deleted.

Examples

The following examples show how to process a DDL job in the PENDING state.

1. Create a table without specifying a primary key. Then, insert duplicate rows into the table. The result indicates that the value 1 for the id column duplicates.

```
mysql> create table test_pending (id int not null, age int) dbpartition by hash(id);
Query OK, 0 rows affected (0.33 sec)
mysql> insert into test_pending values(1,10),(1,20),(2,20),(3,30);
Query OK, 4 rows affected (0.10 sec)
mysql> select * from test_pending order by id;
+------+------+
| id   | age  |
+------+------+
|    1 |   10 |
|    1 |   20 |
|    2 |   20 |
|    3 |   30 |
+------+------+
4 rows in set (0.10 sec)
```

2. Configure a primary key for the table. The result indicates that the id column is incorrectly configured. In this case, the DDL statement fails because the values for the id column are not unique.

```
mysql> alter table test_pending add primary key (id);
ERROR 4636 (HY000): [f5be83373466000][10.81.69.55:3306][ddltest]ERR-CODE: [TDDL-4636][E
RR_DDL_JOB_ERROR] Not all physical operations have been done successfully: expected 9,
but done 8. Caused by: 1062:DDLTEST_1562056402230OYMK_7WW7_0001:Duplicate entry '1' for
key 'PRIMARY' on `test_pending`;.
```

3. Execute the `SHOW FULL DDL` statement to query the job status and the failure cause. The result indicates that the physical DDL statement fails because a physical table contains duplicate column values.

```
mysql> show full ddl\G
*************************** 1. row ***************************
          JOB_ID: 1106733441212637184
   PARENT_JOB_ID: 0
          SERVER: 1:102:10.81.69.55
   OBJECT_SCHEMA: ddltest
     OBJECT_NAME: test_pending
 NEW_OBJECT_NAME:
        JOB_TYPE: ALTER_TABLE
           PHASE: EXECUTE
           STATE: PENDING
        PROGRESS: 77%
      START_TIME: 2019-09-06 17:17:55.002
        END_TIME: 2019-09-06 17:17:55.273
ELAPSED_TIME(MS): 271
        DDL_STMT: alter table test_pending add primary key (id)
          REMARK: ERR-CODE: [TDDL-4636][ERR_DDL_JOB_ERROR] Not all physical operations
have been done successfully: expected 9, but done 8. Caused by: 1062:DDLTEST_1562056402
230OYMK_7WW7_0001:Duplicate entry '1' for key 'PRIMARY' on `test_pending`;.
```

Content in the REMARK field:

- `Not all physical operations have been done successfully: expected 9, but done 8.`: You attempt to execute nine physical DDL statements on the logical table. Eight statements are executed, but one statement fails. This causes a failure in the DDL job. The DDL job enters the PENDING state.
- `Caused by: 1062:DDLTEST_1562056402 230OYMK_7WW7_0001:Duplicate entry '1' for key 'PRIMARY' on 'test_pending';`: This indicates the root cause of the failure. The value 1 for the id column duplicates in the physical table `test_pending`. The physical table is stored in the physical database `DDLTEST_1562056402 230OYMK_7WW7_0001`. Therefore, the id column cannot be used as the primary key.

4. Check the table for errors. The result indicates that the logical table status returned is inconsistent with the actual logical table status.

```
mysql> check table test_pending;
+---------------------------------------+-------+----------+-------------------------
-------------------------------------------------------------------------------+
| TABLE                                 | OP    | MSG_TYPE | MSG_TEXT
|
+---------------------------------------+-------+----------+-------------------------
-------------------------------------------------------------------------------+
| ddltest_1562056402230oymk.test_pending | check | Error    | Table 'DDLTEST_1562056402
230OYMK_7WW7_0001.test_pending' find incorrect columns 'id', please recreate table |
+---------------------------------------+-------+----------+-------------------------
-------------------------------------------------------------------------------+
1 row in set (0.04 sec)
```

5. Check whether subsequent statements in the job can be executed. The result indicates that an error is returned when you execute the DROP TABLE statement.

```
mysql> drop table test_pending;
ERROR 4644 (HY000): [f5beae39d466000][10.81.69.55:3306][ddltest]ERR-CODE: [TDDL-4644][E
RR_PENDING_DDL_JOB_EXISTS] Another DDL job '1106733441212637184' with operation 'ALTER_
TABLE' is pending on ddltest.test_pending in ddltest. Please use SHOW DDL to check it,
and then recover or rollback it using RECOVER DDL or ROLLBACK DDL, or just remove it us
ing REMOVE DDL if you confirm that the pending job can be discarded.
```

6. Use one of the following methods to process the job. For more information about the common methods, see the "How it works" section. The following code shows the effect of each method.
   - Remove duplicates from the table. Then, resume the DDL job to configure a primary key for the table.
     a. Remove duplicates from the table. Before you perform this operation, make sure that you need only one copy of the data. If you want to execute the DELETE statement to remove duplicates, log on to your PolarDB-X 1.0 instance.If an error message is returned when you log on to the PolarDB-X 1.0 instance, you can connect to a backend ApsaraDB RDS for MySQL database based on the error message.

```
mysql> delete from test_pending where id=1 and age=20;
Query OK, 1 row affected (0.07 sec)
mysql> select * from test_pending order by id;
+------+------+
| id   | age  |
+------+------+
|    1 |   10 |
|    2 |   20 |
|    3 |   30 |
+------+------+
3 rows in set (0.02 sec)
```

b. After you remove duplicates, resume the DDL job in the PENDING state. The result indicates that the DDL job resumes, the record of the job failure is cleared, and the primary key is configured for the table.

```
mysql> recover ddl 1106733441212637184;
Query OK, 0 rows affected (1.28 sec)
mysql> show full ddl\G
Empty set (0.00 sec)
mysql> show create table test_pending\G
*************************** 1. row ***************************
       Table: test_pending
Create Table: CREATE TABLE `test_pending` (
  `id` int(11) NOT NULL,
  `age` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `auto_shard_key_id` (`id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`id`)
1 row in set (0.02 sec)
mysql> check table test_pending;
+---------------------------------------+-------+----------+----------+
| TABLE                                 | OP    | MSG_TYPE | MSG_TEXT |
+---------------------------------------+-------+----------+----------+
| ddltest_1562056402230oymk.test_pending | check | status   | OK       |
+---------------------------------------+-------+----------+----------+
1 row in set (0.10 sec)
```

- Delete the failed DDL job, delete the table, and then create the table again. Before you delete the table, make sure that the data on the table is no longer needed.

```
mysql> remove ddl 1106733441212637184;
Query OK, 1 row affected (0.02 sec)
mysql> drop table if exists test_pending;
Query OK, 0 rows affected (0.44 sec)
mysql> show tables like 'test_pending';
Empty set (0.01 sec)
```

4.DDL

4.1. CREATE TABLE

This topic describes the syntax, clauses, parameters, and basic methods for creating a table by executing a data definition language (DDL) statement.

Considerations

- PolarDB-X 1.0 does not allow you to directly create a database by executing a DDL statement. You can log on to the console of the cloud native distributed database to create a database. For more information about how to create a database, see Create a database.
- PolarDB-X 1.0 supports global secondary indexes (GSIs) only when the MySQL version is 5.7 or later and the PolarDB-X 1.0 instance version is 5.4.1 or later. For more information about the basic principles, see Global secondary indexes.

```
CREATE [SHADOW] TABLE [IF NOT EXISTS] tbl_name
    (create_definition, ...)
    [table_options]
    [drds_partition_options]
create_definition:
    col_name column_definition
  | mysql_create_definition
  | [UNIQUE] GLOBAL INDEX index_name [index_type] (index_sharding_col_name,...)
      [global_secondary_index_option]
      [index_option] ...
# GSI-related syntax
global_secondary_index_option:
    [COVERING (col_name,...)]
    [drds_partition_options]
# Clauses for sharding
drds_partition_options:
    DBPARTITION BY db_partition_algorithm
    [TBPARTITION BY table_partition_algorithm [TBPARTITIONS num]]
db_sharding_algorithm:
    HASH([col_name])
  | {YYYYMM|YYYYWEEK|YYYYDD|YYYYMM_OPT|YYYYWEEK_OPT|YYYYDD_OPT}(col_name)
  | UNI_HASH(col_name)
  | RIGHT_SHIFT(col_name, n)
  | RANGE_HASH(col_name, col_name, n)
table_sharding_algorithm:
    HASH(col_name)
  | {MM|DD|WEEK|MMDD|YYYYMM|YYYYWEEK|YYYYDD|YYYYMM_OPT|YYYYWEEK_OPT|YYYYDD_OPT}(col_name)
  | UNI_HASH(col_name)
  | RIGHT_SHIFT(col_name, n)
  | RANGE_HASH(col_name, col_name, n)
# MySQL DDL syntax
index_sharding_col_name:
    col_name [(length)] [ASC | DESC]
index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
index_type:
    USING {BTREE | HASH}
```

Note The PolarDB-X 1.0 DDL syntax is based on the MySQL syntax. The preceding code lists the syntax that is different from the MySQL syntax. For more information about the syntax, see MySQL documentation.

Clauses and parameters for sharding

- `DBPARTITION BY hash(partition_key)`: This clause specifies the database shard key and the database sharding algorithm.
- `TBPARTITION BY { HASH(column) | {MM|DD|WEEK|MMDD|YYYYMM|YYYYWEEK|YYYYDD|YYYYMM_OPT|YYYYWEEK_OPT|YYYYDD_OPT}(column)`: Optional. This clause specifies the method that is used to map data to

a physical table. It is the same as the `DBPARTITION BY` clause by default.

- `TBPARTITIONS num`: Optional. This parameter specifies the number of physical tables in each database. The default value is 1. If no table sharding is required, you do not need to specify this parameter.
- For more information about sharding functions, see Overview.

GSI definition clauses

- `[UNIQUE] GLOBAL`: defines a GSI. UNIQUE GLOBAL indicates a global unique index.
- `index_name`: the name of the index. It is also the name of the index table.
- `index_type`: the type of the local index for a shard key in the index table. For more information about the supported range, see MySQL documentation.
- `index_sharding_col_name,...`: the index columns. This clause contains only all the shard keys of the index table. For more information, see Global secondary indexes.
- `global_secondary_index_option`: the extended syntax for PolarDB-X 1.0PolarDB-X 1.0 GSIs.
  - `COVERING (col_name,...)`: the covering columns. This clause contains all the columns of the index table except the index column. By default, this clause contains the primary key and the shard key of the primary table. For more information, see Global secondary indexes.
  - `drds_partition_options`: the sharding clauses in the index table. For more information, see Clauses and parameters for sharding.
- `index_option`: the attributes of the local index on the shard key of the index table. For more information, see MySQL documentation.

Shadow table clause for full-link stress testing

`SHADOW _test_`: creates a shadow table for full-link stress testing. The table name must be prefixed with .The table name that follows the prefix must be consistent with the name of the associated formal table. In addition, the formal table must be created before the shadow table is created.

Single-database single table

Creates a single-database single table. No sharding is required.

```
CREATE TABLE single_tbl(
 id bigint not null auto_increment,
 name varchar(30),
 primary key(id)
);
```

View the node topology of the logical table. The node topology shows that a single-database single logical table is created in database 0.

```
mysql> show topology from single_tbl;
+------+-----------------------------------------------------------------+------------+
| ID   | GROUP_NAME                                                      | TABLE_NAME |
+------+-----------------------------------------------------------------+------------+
|    0 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | single_tbl |
+------+-----------------------------------------------------------------+------------+
1 row in set (0.01 sec)
```

Database sharding instead of table sharding

Assume that eight database shards are created. Create a table for which only database sharding instead of table sharding is implemented. For the database sharding method, hashing is implemented by using the ID column.

```
CREATE TABLE multi_db_single_tbl(
  id bigint not null auto_increment,
  name varchar(30),
  primary key(id)
) dbpartition by hash(id);
```

View the node topology of the logical table. The node topology shows that one table shard is created in each database shard. This indicates that only database sharding is implemented.

```
mysql> show topology from multi_db_single_tbl;
+------+-----------------------------------------------------------------+----------------
-----+
| ID   | GROUP_NAME                                                      | TABLE_NAME
|
+------+-----------------------------------------------------------------+----------------
-----+
|    0 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | multi_db_single
_tbl |
|    1 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | multi_db_single
_tbl |
|    2 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0002_RDS | multi_db_single
_tbl |
|    3 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0003_RDS | multi_db_single
_tbl |
|    4 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0004_RDS | multi_db_single
_tbl |
|    5 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0005_RDS | multi_db_single
_tbl |
|    6 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0006_RDS | multi_db_single
_tbl |
|    7 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | multi_db_single
_tbl |
+------+-----------------------------------------------------------------+----------------
-----+
8 rows in set (0.01 sec)
```

Sharding

You can implement sharding by using the following sharding methods:

- Use the hash function for sharding
- Use the hash function that has double fields for sharding
- Use dates for sharding

> **Note** In the following examples, assume that eight database shards have been created.

Use the hash function for sharding

Create a table for which both database sharding and table sharding are implemented. Each database shard contains three physical tables. For the database sharding method, hashing is implemented by using the ID column. For the table sharding method, hashing is implemented by using the bid column. You can first perform a hash operation by using the values of the ID column to distribute the table data to multiple database shards. Then, the data in each database shard is distributed to three physical tables by using the hash operation result of the bid column values.

```
CREATE TABLE multi_db_multi_tbl(
 id bigint not null auto_increment,
 bid int,
 name varchar(30),
 primary key(id)
) dbpartition by hash(id) tbpartition by hash(bid) tbpartitions 3;
```

View the node topology of the logical table. The node topology shows that three table shards are created in each database shard.

```
mysql> show topology from multi_db_multi_tbl;
+------+--------------------------------------------------------------------+----------------
-------+
| ID   | GROUP_NAME                                                         | TABLE_NAME
|
+------+--------------------------------------------------------------------+----------------
-------+
|    0 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | multi_db_multi_
tbl_00 |
|    1 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | multi_db_multi_
tbl_01 |
|    2 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | multi_db_multi_
tbl_02 |
|    3 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | multi_db_multi_
tbl_03 |
|    4 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | multi_db_multi_
tbl_04 |
|    5 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | multi_db_multi_
tbl_05 |
|    6 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0002_RDS | multi_db_multi_
tbl_06 |
|    7 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0002_RDS | multi_db_multi_
tbl_07 |
|    8 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0002_RDS | multi_db_multi_
```

```
tbl_08 |
|    9 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0003_RDS | multi_db_multi_
tbl_09 |
|   10 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0003_RDS | multi_db_multi_
tbl_10 |
|   11 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0003_RDS | multi_db_multi_
tbl_11 |
|   12 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0004_RDS | multi_db_multi_
tbl_12 |
|   13 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0004_RDS | multi_db_multi_
tbl_13 |
|   14 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0004_RDS | multi_db_multi_
tbl_14 |
|   15 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0005_RDS | multi_db_multi_
tbl_15 |
|   16 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0005_RDS | multi_db_multi_
tbl_16 |
|   17 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0005_RDS | multi_db_multi_
tbl_17 |
|   18 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0006_RDS | multi_db_multi_
tbl_18 |
|   19 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0006_RDS | multi_db_multi_
tbl_19 |
|   20 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0006_RDS | multi_db_multi_
tbl_20 |
|   21 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | multi_db_multi_
tbl_21 |
|   22 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | multi_db_multi_
tbl_22 |
|   23 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | multi_db_multi_
tbl_23 |
+------+-----------------------------------------------------------------+---------------
-------+
24 rows in set (0.01 sec)
```

View the sharding rule of the logical table. The rule shows that hashing is used for sharding. The shard key for database sharding is ID and the shard key for table sharding is bid.

```
mysql> show rule from multi_db_multi_tbl;
+------+--------------------+-----------+------------------+---------------------+---------
-----------+------------------+---------------------+--------------------+
| ID   | TABLE_NAME         | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTI
TION_COUNT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+--------------------+-----------+------------------+---------------------+---------
-----------+------------------+---------------------+--------------------+
|    0 | multi_db_multi_tbl |         0 | id               | hash                | 8
| bid              | hash                | 3                  |
+------+--------------------+-----------+------------------+---------------------+---------
-----------+------------------+---------------------+--------------------+
1 row in set (0.01 sec)
```

Use the hash function that has double fields for sharding

- The type of the shard key must be the character type or the numeric type.
- Routing method: Calculate a hash value by using the last N digits of a shard key so that the hashing method can be used to complete route computing. The letter N is the third parameter in the function. For example, when the `RANGE_HASH(COL1, COL2, N)` function is used for calculation, COL1 is preferentially selected and then truncated to obtain the last N characters for calculation. If COL1 does not exist, COL2 is selected for calculation.
- Scenarios: Two shard keys are required and only the value of one shard key is used for queries. Assume that eight physical databases have been allocated to PolarDB-X 1.0 of a user and the following scenarios are required for the service:
  - For a specified service, database sharding needs to be implemented for the order table by the buyer ID and the order ID.
  - The condition used during a query is only the buyer ID or the order ID.

In this case, you can execute the following DDL statement to create an order table:

```
create table test_order_tb (
 id bigint not null auto_increment,
 seller_id varchar(30) DEFAULT NULL,
 order_id varchar(30) DEFAULT NULL,
 buyer_id varchar(30) DEFAULT NULL,
 create_time datetime DEFAULT NULL,
 primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by RANGE_HASH(buyer_id, order_id, 10) tbpa
rtition by RANGE_HASH(buyer_id, order_id, 10) tbpartitions 3;
```

> **Note**
>
> - The two shard keys cannot be modified.
> - Data fails to be inserted if the two shard keys point to different database shards or table shards.

Use dates for sharding

You can use a hash function as the sharding algorithm. You can also use the DATE function `MM`, `DD`, `WEEK`, or `MMDD` as the sharding algorithm for table sharding. The following examples show the detailed procedure:

Create a table for which both database sharding and table sharding are implemented. For the database sharding method, hashing is implemented by using the `userId` column. For the table sharding method, the table is sharded by using the `actionDate` column and using seven days as one week. The `WEEK(actionDate)` function calculates `DAY_OF_WEEK`.

For example, if a value in the `actionDate` column is 2017-02-27 that falls on Monday, the value that the `WEEK(actionDate)` function returns is 2. In this case, the record is stored in table shard `2 (2%7 = 2)`. This table shard is located in a database shard and is named `user_log_2`. For another example, if a value in the `actionDate` column is 2017-02-26 that falls on Sunday, the value that the `WEEK(actionDate)` function returns is 1. In this case, the record is stored in table shard `1 (1%7 = 1)`. This table shard is located in a database shard and is named `user_log_1`.

```
CREATE TABLE user_log(
 userId int,
 name varchar(30),
 operation varchar(30),
 actionDate DATE
) dbpartition by hash(userId) tbpartition by WEEK(actionDate) tbpartitions 7;
```

View the node topology of the logical table. The node topology shows that seven table shards (one week has seven days) are created in each database shard.

```
mysql> show topology from user_log;
+------+-------------------------------------------------------------------+------------+
| ID   | GROUP_NAME                                                        | TABLE_NAME |
+------+-------------------------------------------------------------------+------------+
|    0 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log_0 |
|    1 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log_1 |
|    2 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log_2 |
|    3 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log_3 |
|    4 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log_4 |
|    5 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log_5 |
|    6 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log_6 |
|    7 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log_0 |
|    8 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log_1 |
|    9 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log_2 |
|   10 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log_3 |
|   11 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log_4 |
|   12 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log_5 |
|   13 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log_6 |
...
|   49 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log_0 |
|   50 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log_1 |
|   51 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log_2 |
|   52 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log_3 |
|   53 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log_4 |
|   54 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log_5 |
|   55 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log_6 |
+------+-------------------------------------------------------------------+------------+
56 rows in set (0.01 sec)
```

Note An ellipsis (...) is used to omit some data because the returned result is long.

View the sharding rule of the logical table. The rule shows that the database sharding method is hashing and the shard key for database sharding is `userId`. For the table sharding method, the table is sharded by using the time function `WEEK` and the shard key for table sharding is `actionDate`.

```
mysql> show rule from user_log;
+------+------------+-----------+-----------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
| ID   | TABLE_NAME | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COU
NT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+------------+-----------+-----------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
|    0 | user_log   |         0 | userId           | hash                | 8
| actionDate       | week                | 7                  |
+------+------------+-----------+-----------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
1 row in set (0.00 sec)
```

View the physical database shard and its physical table to which the SQL statement is routed when the parameters of the database shard key and the table shard key are specified.

Create a table for which both database sharding and table sharding are implemented. For the database sharding method, hashing is implemented by using the `userId` column. For the table sharding method, the table is sharded by using the `actionDate` column and using 12 months as one year. The `MM(actionDate)` function calculates `MONTH_OF_YEAR`.

For example, if a value in the `actionDate` column is 2017-02-27, the value that the `MM(actionDate)` function returns is 02. In this case, the record is stored in table shard `02 (02%12 = 02)`. This table shard is located in a database shard and is named `user_log_02`. For another example, if the value in the `actionDate` column is 2016-12-27, the value that the `MM(actionDate)` function returns is 12. In this case, the record is stored in table shard `00 (12%12 = 00)`. This table shard is located in a database shard and is named `user_log_00`.

```
CREATE TABLE user_log2(
 userId int,
 name varchar(30),
 operation varchar(30),
 actionDate DATE
) dbpartition by hash(userId) tbpartition by MM(actionDate) tbpartitions 12;
```

View the node topology of the logical table. The node topology shows that 12 table shards (one year has 12 months) are created in each database shard.

```
mysql> show topology from user_log2;
+------+-------------------------------------------------------------------+--------------+
| ID   | GROUP_NAME                                                        | TABLE_NAME   |
+------+-------------------------------------------------------------------+--------------+
|    0 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_00 |
|    1 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_01 |
|    2 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_02 |
|    3 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_03 |
|    4 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_04 |
|    5 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_05 |
|    6 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_06 |
|    7 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_07 |
|    8 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_08 |
|    9 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_09 |
|   10 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_10 |
|   11 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log2_11 |
|   12 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_00 |
|   13 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_01 |
|   14 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_02 |
|   15 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_03 |
|   16 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_04 |
|   17 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_05 |
|   18 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_06 |
|   19 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_07 |
|   20 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_08 |
|   21 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_09 |
|   22 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_10 |
|   23 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0001_RDS | user_log2_11 |
...
|   84 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_00 |
|   85 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_01 |
|   86 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_02 |
|   87 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_03 |
|   88 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_04 |
|   89 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_05 |
|   90 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_06 |
|   91 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_07 |
|   92 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_08 |
|   93 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_09 |
|   94 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_10 |
|   95 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log2_11 |
+------+-------------------------------------------------------------------+--------------+
96 rows in set (0.02 sec)
```

Note An ellipsis (...) is used to omit some data because the returned result is long.

View the sharding rule of the logical table. The rule shows that the database sharding method is hashing and the shard key for database sharding is `userId`. For the table sharding method, the table is sharded by using the time function `MM` and the shard key for table sharding is `actionDate`.

```
mysql> show rule from user_log2;
+------+------------+-----------+-----------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
| ID   | TABLE_NAME | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COU
NT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+------------+-----------+-----------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
|    0 | user_log2  |         0 | userId          | hash               | 8
| actionDate       | mm                 | 12                 |
+------+------------+-----------+-----------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
1 row in set (0.00 sec)
```

Create a table for which both database sharding and table sharding are implemented. For the database sharding method, hashing is implemented by using the `userId` column. For the table sharding method, the table is sharded by using 31 days as a month. The `DD(actionDate)` function calculates `DAY_OF_MONTH`.

For example, if a value in the `actionDate` column is 2017-02-27, the value that the `DD(actionDate)` function returns is 27. In this case, the record is stored in table shard `27 (27%31 = 27)`. This table shard is located in a database shard and is named `user_log_27`.

```
CREATE TABLE user_log3(
 userId int,
 name varchar(30),
 operation varchar(30),
 actionDate DATE
) dbpartition by hash(userId) tbpartition by DD(actionDate) tbpartitions 31;
```

View the node topology of the logical table. The node topology shows that 31 table shards (each month has 31 days) are created in each database shard.

```
mysql> show topology from user_log3;
+------+-------------------------------------------------------------------+--------------+
| ID   | GROUP_NAME                                                        | TABLE_NAME   |
+------+-------------------------------------------------------------------+--------------+
|    0 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_00 |
|    1 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_01 |
|    2 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_02 |
|    3 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_03 |
|    4 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_04 |
|    5 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_05 |
|    6 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_06 |
|    7 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_07 |
|    8 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_08 |
|    9 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_09 |
|   10 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_10 |
|   11 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_11 |
|   12 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_12 |
|   13 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_13 |
|   14 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_14 |
|   15 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_15 |
|   16 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_16 |
|   17 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_17 |
|   18 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_18 |
|   19 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_19 |
|   20 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_20 |
|   21 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_21 |
|   22 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_22 |
|   23 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_23 |
|   24 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_24 |
|   25 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_25 |
|   26 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_26 |
|   27 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_27 |
|   28 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_28 |
|   29 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_29 |
|   30 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log3_30 |
...
|  237 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_20 |
|  238 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_21 |
|  239 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_22 |
|  240 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_23 |
|  241 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_24 |
|  242 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_25 |
|  243 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_26 |
|  244 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_27 |
|  245 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_28 |
|  246 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_29 |
|  247 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log3_30 |
+------+-------------------------------------------------------------------+--------------+
248 rows in set (0.01 sec)
```

Note An ellipsis (...) is used to omit some data because the returned result is long.

View the sharding rule of the logical table. The rule shows that the database sharding method is hashing and the shard key for database sharding is `userId`. For the table sharding method, the table is sharded by using the time function `DD` and the shard key for table sharding is `actionDate`.

```
mysql> show rule from user_log3;
+------+------------+-----------+------------------+---------------------+-----------------
---+------------------+---------------------+--------------------+
| ID   | TABLE_NAME | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COU
NT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+------------+-----------+------------------+---------------------+-----------------
---+------------------+---------------------+--------------------+
|    0 | user_log3  |         0 | userId           | hash                | 8
| actionDate       | dd                  | 31                  |
+------+------------+-----------+------------------+---------------------+-----------------
---+------------------+---------------------+--------------------+
1 row in set (0.01 sec)
```

Create a table for which both database sharding and table sharding are implemented. For the database sharding method, hashing is implemented by using the `userId` column. For the table sharding method, the table is sharded by using 365 days as one year and the table data is routed to 365 physical tables. The `MMDD(actionDate) tbpartitions 365` function calculates `DAY_OF_YEAR % 365`.

For example, if a value in the `actionDate` column is 2017-02-27, the value that the `MMDD(actionDate)` function returns is 58. In this case, the record is stored in table shard 58. This table shard is located in a database shard and is named `user_log_58`.

```
CREATE TABLE user_log4(
 userId int,
 name varchar(30),
 operation varchar(30),
 actionDate DATE
) dbpartition by hash(userId) tbpartition by MMDD(actionDate) tbpartitions 365;
```

View the node topology of the logical table. The node topology shows that 365 table shards (each year has 365 days) are created in each database shard.

```
mysql> show topology from user_log4;
+------+-------------------------------------------------------------------+---------------+
| ID   | GROUP_NAME                                                        | TABLE_NAME    |
+------+-------------------------------------------------------------------+---------------+
...
| 2896 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_341 |
| 2897 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_342 |
| 2898 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_343 |
| 2899 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_344 |
| 2900 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_345 |
| 2901 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_346 |
| 2902 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_347 |
| 2903 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_348 |
| 2904 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_349 |
| 2905 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_350 |
| 2906 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_351 |
| 2907 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_352 |
| 2908 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_353 |
| 2909 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_354 |
| 2910 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_355 |
| 2911 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_356 |
| 2912 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_357 |
| 2913 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_358 |
| 2914 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_359 |
| 2915 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_360 |
| 2916 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_361 |
| 2917 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_362 |
| 2918 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_363 |
| 2919 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log4_364 |
+------+-------------------------------------------------------------------+---------------+
2920 rows in set (0.07 sec)
```

Note An ellipsis (...) is used to omit some data because the returned result is long.

View the sharding rule of the logical table. The rule shows that the database sharding method is hashing and the shard key for database sharding is `userId`. For the table sharding method, the table is sharded by using the time function `MMDD` and the shard key for table sharding is `actionDate`.

```
mysql> show rule from user_log4;
+------+------------+-----------+-----------------+---------------------+-----------------
---+------------------+---------------------+--------------------+
| ID   | TABLE_NAME | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COU
NT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+------------+-----------+-----------------+---------------------+-----------------
---+------------------+---------------------+--------------------+
|    0 | user_log4  |         0 | userId          | hash                | 8
| actionDate       | mmdd                | 365                |
+------+------------+-----------+-----------------+---------------------+-----------------
---+------------------+---------------------+--------------------+
1 row in set (0.02 sec)
```

Create a table for which both database sharding and table sharding are implemented. For the database sharding method, hashing is implemented by using the `userId` column. For the table sharding method, the table is sharded by using 365 days as one year and the table data is routed to 10 physical tables. The `MMDD(actionDate) tbpartitions 10` function calculates `DAY_OF_YEAR % 10`.

```
CREATE TABLE user_log5(
 userId int,
 name varchar(30),
 operation varchar(30),
 actionDate DATE
) dbpartition by hash(userId) tbpartition by MMDD(actionDate) tbpartitions 10;
```

View the node topology of the logical table. The node topology shows that 10 table shards are created in each database shard. The logical table is sharded by using 365 days as one year and the table data is routed to 10 physical tables.

```
mysql> show topology from user_log5;
+------+--------------------------------------------------------------------+--------------+
| ID   | GROUP_NAME                                                         | TABLE_NAME   |
+------+--------------------------------------------------------------------+--------------+
|    0 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_00 |
|    1 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_01 |
|    2 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_02 |
|    3 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_03 |
|    4 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_04 |
|    5 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_05 |
|    6 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_06 |
|    7 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_07 |
|    8 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_08 |
|    9 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0000_RDS | user_log5_09 |
...
|   70 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_00 |
|   71 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_01 |
|   72 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_02 |
|   73 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_03 |
|   74 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_04 |
|   75 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_05 |
|   76 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_06 |
|   77 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_07 |
|   78 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_08 |
|   79 | SANGUAN_TEST_123_1488766060743ACTJSANGUAN_TEST_123_WVVP_0007_RDS | user_log5_09 |
+------+--------------------------------------------------------------------+--------------+
80 rows in set (0.02 sec)
```

Note An ellipsis (...) is used to omit some data because the returned result is long.

View the sharding rule of the logical table. The rule shows that the database sharding method is hashing and the shard key for database sharding is `userId`. For the table sharding method, the table is sharded by using the time function `MMDD` and the table data is routed to 10 physical tables. In addition, the shard key for table sharding is `actionDate`.

```
mysql> show rule from user_log5;
+------+------------+-----------+------------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
| ID   | TABLE_NAME | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COU
NT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+------------+-----------+------------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
|    0 | user_log5  |         0 | userId           | hash               | 8
| actionDate       | mmdd               | 10                 |
+------+------------+-----------+------------------+--------------------+-----------------
---+------------------+--------------------+--------------------+
1 row in set (0.01 sec)
```

Use the primary key as the shard key

When you do not specify a shard key for the sharding algorithm, the system uses the primary key as the shard field by default. The following examples illustrate how to use the primary key as the database shard key and the table shard key.

- Use the primary key as the database shard key

```
CREATE TABLE prmkey_tbl(
 id bigint not null auto_increment,
 name varchar(30),
 primary key(id)
) dbpartition by hash();
```

- Use the primary key as the shard key

```
CREATE TABLE prmkey_multi_tbl(
 id bigint not null auto_increment,
 name varchar(30),
 primary key(id)
) dbpartition by hash() tbpartition by hash() tbpartitions 3;
```

Other table creation attributes of MySQL

When you implement sharding, you can also specify other table creation attributes of MySQL, as shown in the following example:

```
CREATE TABLE multi_db_multi_tbl(
  id bigint not null auto_increment,
  name varchar(30),
  primary key(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(id) tbpartition by hash(id) tbpart
itions 3;
```

GSI

This section describes how to define a GSI when you create a table.

- Define a GSI
- Define a globally unique index

> **Note** In the following examples, assume that eight database shards have been created.

Define a GSI

Examples

```
CREATE TABLE t_order (
 `id` bigint(11) NOT NULL AUTO_INCREMENT,
 `order_id` varchar(20) DEFAULT NULL,
 `buyer_id` varchar(20) DEFAULT NULL,
 `seller_id` varchar(20) DEFAULT NULL,
 `order_snapshot` longtext DEFAULT NULL,
 `order_detail` longtext DEFAULT NULL,
 PRIMARY KEY (`id`),
 GLOBAL INDEX `g_i_seller`(`seller_id`) dbpartition by hash(`seller_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
```

where:

- Primary table: `t_order` is the primary table for which database sharding instead of table sharding is implemented. For the database sharding method, hashing is implemented by using the `order_id` column.
- Index table: `g_i_seller` is the index table for which database sharding instead of table sharding is implemented. For the database sharding method, hashing is implemented by using the `seller_id` column. No covering column is specified.
- GSI definition clause: `GLOBAL INDEX `g_i_seller`(`seller_id`) dbpartition by hash(`seller_id`)`.

Execute `SHOW INDEX` to view index information, such as the local index on the shard key `order_id` and GSIs on `seller_id`, `id`, and `order_id`. `seller_id` is the shard key of the index table, and `id` and `order_id` are the default covering columns (the primary key and the shard key of the primary table).

```
mysql> show index from t_order;
+---------+------------+------------------------+--------------+-------------+-----------+
-------------+----------+--------+------+------------+----------+---------------+
| TABLE   | NON_UNIQUE | KEY_NAME               | SEQ_IN_INDEX | COLUMN_NAME | COLLATION |
CARDINALITY | SUB_PART | PACKED | NULL | INDEX_TYPE | COMMENT  | INDEX_COMMENT |
+---------+------------+------------------------+--------------+-------------+-----------+
-------------+----------+--------+------+------------+----------+---------------+
| t_order |          0 | PRIMARY                |            1 | id          | A         |
0 |     NULL | NULL   |      | BTREE      |          |               |
| t_order |          1 | auto_shard_key_order_id |            1 | order_id    | A         |
0 |     NULL | NULL   | YES  | BTREE      |          |               |
| t_order |          1 | g_i_seller             |            1 | seller_id   | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | INDEX    |               |
| t_order |          1 | g_i_seller             |            2 | id          | NULL      |
0 |     NULL | NULL   |      | GLOBAL     | COVERING |               |
| t_order |          1 | g_i_seller             |            3 | order_id    | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
+---------+------------+------------------------+--------------+-------------+-----------+
-------------+----------+--------+------+------------+----------+---------------+
```

You can separately view the GSI information by executing `SHOW GLOBAL INDEX`. For more information, see SHOW GLOBAL INDEX.

```
 mysql> show global index from t_order;
+--------+---------+------------+------------+-------------+----------------+------------+-
----------------+---------------------+--------------------+------------------+-----------
----------+--------------------+--------+
| SCHEMA | TABLE   | NON_UNIQUE | KEY_NAME   | INDEX_NAMES | COVERING_NAMES | INDEX_TYPE |
DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COUNT | TB_PARTITION_KEY | TB_PARTITI
ON_POLICY | TB_PARTITION_COUNT | STATUS |
+--------+---------+------------+------------+-------------+----------------+------------+-
----------------+---------------------+--------------------+------------------+-----------
----------+--------------------+--------+
| d7     | t_order | 1          | g_i_seller | seller_id   | id, order_id   | NULL       |
seller_id        | HASH               | 8                  |                  | NULL
| NULL               | PUBLIC |
+--------+---------+------------+------------+-------------+----------------+------------+-
----------------+---------------------+--------------------+------------------+-----------
----------+--------------------+--------+
```

View the schema of the index table. The index table contains the primary key of the primary table, shard key, and default covering columns. The `AUTO_INCREMENT` attribute is removed from the primary key column and the local index is removed from the primary table.

```
mysql> show create table g_i_seller;
+------------+-----------------------------------------------------------------+
| Table      | Create Table                                                    |
+------------+-----------------------------------------------------------------+
| g_i_seller | CREATE TABLE `g_i_seller` (
 `id` bigint(11) NOT NULL,
 `order_id` varchar(20) DEFAULT NULL,
 `seller_id` varchar(20) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `auto_shard_key_seller_id` (`seller_id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`seller_id`) |
+------------+-----------------------------------------------------------------+
```

Define a globally unique index

```
CREATE TABLE t_order (
 `id` bigint(11) NOT NULL AUTO_INCREMENT,
 `order_id` varchar(20) DEFAULT NULL,
 `buyer_id` varchar(20) DEFAULT NULL,
 `seller_id` varchar(20) DEFAULT NULL,
 `order_snapshot` longtext DEFAULT NULL,
 `order_detail` longtext DEFAULT NULL,
 PRIMARY KEY (`id`),
 UNIQUE GLOBAL INDEX `g_i_buyer`(`buyer_id`) COVERING(`seller_id`, `order_snapshot`)
   dbpartition by hash(`buyer_id`) tbpartition by hash(`buyer_id`) tbpartitions 3
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
```

where:

- Primary table: `t_order` is the primary table for which database sharding instead of table sharding is implemented. For the database sharding method, hashing is implemented by using the `order_id` column.
- Index table: `g_i_seller` is the index table for which both database sharding and table sharding are implemented. For the database sharding and table sharding methods, hashing is implemented by using the `buyer_id` column. The covering columns are `seller_id` and `order_snapshot`.
- Index definition clause: `UNIQUE GLOBAL INDEX `g_i_buyer`(`buyer_id`) COVERING(`seller_id`, `order_snapshot`) dbpartition by hash(`buyer_id`) tbpartition by hash(`buyer_id`) tbpartitions 3`.

Execute `SHOW INDEX` to view index information, such as the local index on the shard key `order_id` and GSIs on `buyer_id`, `id`, `order_id`, `seller_id`, and `order_snapshot`. `buyer_id` is the shard key of the index table. `id` and `order_id` are the default covering columns (the primary key and the shard key of the primary table). `seller_id` and `order_snapshot` are the explicitly specified covering columns.

```
mysql> show index from t_order;
+--------------+------------+------------------------+--------------+----------------+----
-------+-------------+----------+--------+------+------------+----------+---------------+
| TABLE        | NON_UNIQUE | KEY_NAME               | SEQ_IN_INDEX | COLUMN_NAME    | COL
LATION | CARDINALITY | SUB_PART | PACKED | NULL | INDEX_TYPE | COMMENT  | INDEX_COMMENT |
+--------------+------------+------------------------+--------------+----------------+----
-------+-------------+----------+--------+------+------------+----------+---------------+
| t_order_dthb |          0 | PRIMARY                |            1 | id             | A
|           0 |     NULL | NULL   |      | BTREE      |          |               |
| t_order_dthb |          1 | auto_shard_key_order_id |            1 | order_id       | A
|           0 |     NULL | NULL   | YES  | BTREE      |          |               |
| t_order      |          0 | g_i_buyer              |            1 | buyer_id       | NUL
L      |           0 |     NULL | NULL   | YES  | GLOBAL     | INDEX    |               |
| t_order      |          1 | g_i_buyer              |            2 | id             | NUL
L      |           0 |     NULL | NULL   |      | GLOBAL     | COVERING |               |
| t_order      |          1 | g_i_buyer              |            3 | order_id       | NUL
L      |           0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
| t_order      |          1 | g_i_buyer              |            4 | seller_id      | NUL
L      |           0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
| t_order      |          1 | g_i_buyer              |            5 | order_snapshot | NUL
L      |           0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
+--------------+------------+------------------------+--------------+----------------+----
-------+-------------+----------+--------+------+------------+----------+---------------+
```

You can separately view the GSI information by executing `SHOW GLOBAL INDEX`. For more information, see SHOW GLOBAL INDEX.

```
mysql> show global index from t_order;
+--------+---------+------------+-----------+-------------+--------------------------------
---------+------------+-----------------+--------------------+--------------------+------
-----------+---------------------+-------------------+--------+
| SCHEMA | TABLE   | NON_UNIQUE | KEY_NAME  | INDEX_NAMES | COVERING_NAMES
| INDEX_TYPE | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COUNT | TB_PARTITION_K
EY | TB_PARTITION_POLICY | TB_PARTITION_COUNT | STATUS |
+--------+---------+------------+-----------+-------------+--------------------------------
---------+------------+-----------------+--------------------+--------------------+------
-----------+---------------------+-------------------+--------+
| d7     | t_order | 0          | g_i_buyer | buyer_id    | id, order_id, seller_id, order_
snapshot | NULL       | buyer_id         | HASH               | 8                  | buyer
_id         | HASH                | 3                  | PUBLIC |
+--------+---------+------------+-----------+-------------+--------------------------------
---------+------------+-----------------+--------------------+--------------------+------
-----------+---------------------+-------------------+--------+
```

View the schema of the index table. The index table contains the primary key of the primary table, shard key, default covering columns, and the covering columns that are specified in the GSI definition. The `AUTO_INCREMENT` attribute is removed from the primary key column. The local index is removed from the primary table. By default, a data table is created for global unique indexes to support global uniqueness.

```
mysql> show create table g_i_buyer;
+-----------+-------------------------------------------------------------------------------------
--------------------------+
| Table     | Create Table
|
+-----------+-------------------------------------------------------------------------------------
--------------------------+
| g_i_buyer | CREATE TABLE `g_i_buyer` (
  `id` bigint(11) NOT NULL,
  `order_id` varchar(20) DEFAULT NULL,
  `buyer_id` varchar(20) DEFAULT NULL,
  `seller_id` varchar(20) DEFAULT NULL,
  `order_snapshot` longtext,
  PRIMARY KEY (`id`),
  UNIQUE KEY `auto_shard_key_buyer_id` (`buyer_id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`buyer_id`) tbpartition by hash(`b
uyer_id`) tbpartitions 3 |
+-----------+-------------------------------------------------------------------------------------
--------------------------+
```

4.2. DROP TABLE

This topic describes how to use the DROP TABLE statement to delete a specified table.

Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

Note Data tables in PolarDB-X 1.0 can be deleted by using the same syntax that is used to delete tables in MySQL databases. The system automatically processes or deletes the related physical tables. For more information, see DROP TABLE statement.

Note

- If you execute the DROP TABLE statement to delete a table that contains a global secondary index, the index table and the primary table are both deleted.
- To delete only the index table, execute the ALTER TABLE DROP INDEX statement instead of the DROP TABLE statement. For more information, see ALTER TABLE.

Examples

The following example shows how to delete the table named `user_log`:

```
DROP TABLE user_log;
```

4.3. ALTER TABLE

You can execute the ALTER TABLE statement to modify the schema of a table. For example, you can add a column, create an index, or change a data type.

Note

- You cannot execute the ALTER TABLE statement to change a shard key.
- If you need to execute the ALTER TABLE statement on a table that contains a global secondary index (GSI), use MySQL 5.7 or later and PolarDB-X 1.0 5.4.1 or later.

Modify a standard table

> **Note** PolarDB-X 1.0 If you execute the ALTER TABLE statement to modify the schema of a standard table, the syntax of this statement in DRDS is the same as that in open source MySQL. For more information, see ALTER TABLE statement.

Syntax

```
ALTER [ONLINE|OFFLINE] [IGNORE] TABLE tbl_name
    [alter_specification [, alter_specification] ...]
    [partition_options]
```

Examples

- Add a column

  Add the idcard column to the user_log table. You can use the following sample code:

  ```
  ALTER TABLE user_log ADD COLUMN idcard varchar(30);
  ```

- Create a local index

  Create an index named idcard_idx on the idcard column in the user_log table. You can use the following sample code:

  ```
  ALTER TABLE user_log ADD INDEX idcard_idx (idcard);
  ```

- Rename a local index

  Rename the idcard_idx index in the user_log table as idcard_idx_new. You can use the following sample code:

  ```
  ALTER TABLE user_log RENAME INDEX `idcard_idx` TO `idcard_idx_new`;
  ```

- Delete a local index

  Delete the idcard_idx index from the user_log table. You can use the following sample code:

  ```
  ALTER TABLE user_log DROP INDEX idcard_idx;
  ```

- Modify a column

  Change the length of the idcard column in the user_log table from 30 characters to 40 characters. The values for the idcard column are of the VARCHAR type. You can use the following sample code:

  ```
  ALTER TABLE user_log MODIFY COLUMN idcard varchar(40);
  ```

Modify a table that contains a GSI

Modify a column

When you execute the ALTER TABLE statement to modify a column in a table that contains a GSI, the syntax of this statement is the same as that you use to modify a column in a standard table. We recommend that you are familiar with the limits. For more information about the limits, see Notes for executing the ALTER TABLE statement.

Modify an index

Syntax

```
ALTER TABLE tbl_name
    alter_specification # If you execute the ALTER TABLE statement to modify a GSI, use the
alter_specification option once.
alter_specification:
  | ADD GLOBAL {INDEX|KEY} index_name # Explicitly specify the name of a GSI.
      [index_type] (index_sharding_col_name,...)
      global_secondary_index_option
      [index_option] ...
  | ADD [CONSTRAINT [symbol]] UNIQUE GLOBAL
      [INDEX|KEY] index_name # Explicitly specify the name of a GSI.
      [index_type] (index_sharding_col_name,...)
      global_secondary_index_option
      [index_option] ...
  | DROP {INDEX|KEY} index_name
  | RENAME {INDEX|KEY} old_index_name TO new_index_name
global_secondary_index_option:
    [COVERING (col_name,...)] # Covering Index
    drds_partition_options # Specify one or more columns that are contained in index_shardi
ng_col_name.
# Specify a sharding method for an index table.
drds_partition_options:
    DBPARTITION BY db_sharding_algorithm
    [TBPARTITION BY {table_sharding_algorithm} [TBPARTITIONS num]]
db_sharding_algorithm:
    HASH([col_name])
  | {YYYYMM|YYYYWEEK|YYYYDD|YYYYMM_OPT|YYYYWEEK_OPT|YYYYDD_OPT}(col_name)
  | UNI_HASH(col_name)
  | RIGHT_SHIFT(col_name, n)
  | RANGE_HASH(col_name, col_name, n)
table_sharding_algorithm:
    HASH(col_name)
  | {MM|DD|WEEK|MMDD|YYYYMM|YYYYWEEK|YYYYDD|YYYYMM_OPT|YYYYWEEK_OPT|YYYYDD_OPT}(col_name)
  | UNI_HASH(col_name)
  | RIGHT_SHIFT(col_name, n)
  | RANGE_HASH(col_name, col_name, n)
# The following sample code uses the DDL syntax that is supported by the MySQL engine:
index_sharding_col_name:
    col_name [(length)] [ASC | DESC]
index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
index_type:
    USING {BTREE | HASH}
```

After you create a table, you can use `ALTER TABLE ADD GLOBAL INDEX` to create a GSI. Compared with the syntax for MySQL, the syntax for DRDS introduces the GLOBAL keyword to specify that you create a GSI.

You can also use `ALTER TABLE { DROP | RENAME } INDEX` to modify a GSI. If you create a GSI after a table is created, we recommend that you are familiar with the limits. For more information about the limits, see Notes for using GSIs.

For more information about the clauses that are used to define GSIs, see CREATE TABLE.

Examples

The following examples show how to create a unique GSI after a table is created.

- Create a GSI.

```
# Create a table.
CREATE TABLE t_order (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `order_id` varchar(20) DEFAULT NULL,
  `buyer_id` varchar(20) DEFAULT NULL,
  `seller_id` varchar(20) DEFAULT NULL,
  `order_snapshot` longtext DEFAULT NULL,
  `order_detail` longtext DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `l_i_order` (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
# Create a GSI.
ALTER TABLE t_order ADD UNIQUE GLOBAL INDEX `g_i_buyer` (`buyer_id`) COVERING (`order_sna
pshot`) dbpartition by hash(`buyer_id`);
```

  - Primary table: The data on the primary table t_order is partitioned into database shards but not further partitioned into table shards. The database uses hash sharding based on the order_id column.
  - Index table: The data on the index table g_i_buyer is partitioned into database shards but not further partitioned into table shards. The database uses hash sharding based on the buyer_id column. order_snapshot is the covering column that you specify.
  - Clause used to define the GSI: `GLOBAL INDEX `g_i_seller` ON t_order (`seller_id`) dbpartitio
n by hash(`seller_id`)` .

- Execute the `SHOW INDEX` statement to query index information. For example, you can query local indexes on the shard key order_id and the GSIs on the columns buyer_id, id, order_id, and order_snapshot. For the index table, the buyer_id column is the shard key, the id column is the primary key, and the order_id column is the shard key. The id and order_id columns are the default covering columns. order_snapshot is the covering column that you explicitly specify.

```
mysql> show index from t_order;
+---------+------------+-----------+--------------+----------------+-----------+---------
----+----------+--------+------+------------+----------+---------------+
| TABLE   | NON_UNIQUE | KEY_NAME  | SEQ_IN_INDEX | COLUMN_NAME    | COLLATION | CARDINAL
ITY | SUB_PART | PACKED | NULL | INDEX_TYPE | COMMENT  | INDEX_COMMENT |
+---------+------------+-----------+--------------+----------------+-----------+---------
----+----------+--------+------+------------+----------+---------------+
| t_order |          0 | PRIMARY   |            1 | id             | A         |
0 |     NULL | NULL   |      | BTREE      |          |               |
| t_order |          1 | l_i_order |            1 | order_id       | A         |
0 |     NULL | NULL   | YES  | BTREE      |          |               |
| t_order |          0 | g_i_buyer |            1 | buyer_id       | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | INDEX    |               |
| t_order |          1 | g_i_buyer |            2 | id             | NULL      |
0 |     NULL | NULL   |      | GLOBAL     | COVERING |               |
| t_order |          1 | g_i_buyer |            3 | order_id       | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
| t_order |          1 | g_i_buyer |            4 | order_snapshot | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
+---------+------------+-----------+--------------+----------------+-----------+---------
----+----------+--------+------+------------+----------+---------------+
```

- Execute the `SHOW GLOBAL INDEX` statement to query GSI information. For more information, see SHOW GLOBAL INDEX.

```
mysql> show global index from t_order;
+---------------------+---------+------------+-----------+-------------+-----------------
-------------+------------+------------------+---------------------+--------------------+
------------------+---------------------+--------------------+--------+
| SCHEMA              | TABLE   | NON_UNIQUE | KEY_NAME  | INDEX_NAMES | COVERING_NAMES
| INDEX_TYPE | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COUNT | TB_PARTITION
_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT | STATUS |
+---------------------+---------+------------+-----------+-------------+-----------------
-------------+------------+------------------+---------------------+--------------------+
------------------+---------------------+--------------------+--------+
| ZZY3_DRDS_LOCAL_APP | t_order | 0          | g_i_buyer | buyer_id    | id, order_id, or
der_snapshot | NULL       | buyer_id         | HASH                | 4                  |
| NULL                | NULL               | PUBLIC |
+---------------------+---------+------------+-----------+-------------+-----------------
-------------+------------+------------------+---------------------+--------------------+
------------------+---------------------+--------------------+--------+
```

- View the schema of the index table. The index table contains the primary key of the primary table, the database shard key and table shard key, the default covering columns, and the custom covering columns. The AUTO_INCREMENT attribute is removed from the primary key. The local index is removed from the primary table. By default, GSIs are created on all the shard keys of the index table and each GSI is globally unique.

```
mysql> show create table g_i_buyer;
+-----------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| Table     | Create Table
|
+-----------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| g_i_buyer | CREATE TABLE `g_i_buyer` (
  `id` bigint(11) NOT NULL,
  `order_id` varchar(20) DEFAULT NULL,
  `buyer_id` varchar(20) DEFAULT NULL,
  `order_snapshot` longtext,
  PRIMARY KEY (`id`),
  UNIQUE KEY `auto_shard_key_buyer_id` (`buyer_id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`buyer_id`) |
+-----------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
```

- Delete a GSI.

  Delete a GSI named g_i_seller. In this case, the index table named g_i_seller is also deleted.

  ```
  ALTER TABLE `t_order` DROP INDEX `g_i_seller`;
  ```

- Rename a GSI.

  By default, you cannot rename a GSI.

4.4. TRUNCATE TABLE

You can execute the TRUNCATE TABLE statement to clear data from a table.

Syntax

```
TRUNCATE [TABLE] tbl_name
```

For more information about the syntax, see TRUNCATE TABLE statement.

Note

Before you execute the TRUNCATE TABLE statement, ensure that you have the DROP permission.

4.5. RENAME TABLE

This topic describes how to use the RENAME INDEX statement to rename a table.

Syntax

```
RENAME TABLE
    tbl_name TO new_tbl_name
```

Note

- PolarDB-X 1.0 does not allow you to execute the RENAME TABLE statement to rename multiple tables at a time.
- The system does not allow you to rename a table that contains a global secondary index for stability and performance considerations.
- You cannot use only the RENAME INDEX statement to rename an index table. If you need to rename an index table, we recommend that you use the RENAME INDEX statement. For more information, see ALTER TABLE.

4.6. CREATE INDEX

This topic describes how to use the CREATE INDEX statement to create a local secondary index (LSI) or a global secondary index (GSI).

Note

To execute the ALTER statement on a table that contains a GSI, ensure that the MySQL version is 5.7 or later and the PolarDB-X 1.0 version is V5.4.1 or later.

LSI

For more information, see CREATE INDEX statement.

GSI

Syntax

```
CREATE [UNIQUE]
    GLOBAL INDEX index_name [index_type]
    ON tbl_name (index_sharding_col_name,...)
    global_secondary_index_option
    [index_option]
    [algorithm_option | lock_option] ...
# GSI-specific syntax. For more information, see CREATE TABLE statement in MySQL.
global_secondary_index_option:
    [COVERING (col_name,...)]
    drds_partition_options
# Clauses for sharding. For more information, see CREATE TABLE statement in MySQL.
drds_partition_options:
    DBPARTITION BY db_sharding_algorithm
    [TBPARTITION BY {table_sharding_algorithm} [TBPARTITIONS num]]
db_sharding_algorithm:
    HASH([col_name])
  | {YYYYMM|YYYYWEEK|YYYYDD|YYYYMM_OPT|YYYYWEEK_OPT|YYYYDD_OPT}(col_name)
  | UNI_HASH(col_name)
  | RIGHT_SHIFT(col_name, n)
  | RANGE_HASH(col_name, col_name, n)
table_sharding_algorithm:
    HASH(col_name)
  | {MM|DD|WEEK|MMDD|YYYYMM|YYYYWEEK|YYYYDD|YYYYMM_OPT|YYYYWEEK_OPT|YYYYDD_OPT}(col_name)
  | UNI_HASH(col_name)
  | RIGHT_SHIFT(col_name, n)
  | RANGE_HASH(col_name, col_name, n)
 # The following sample code uses the DDL syntax that is supported by the MySQL engine:
index_sharding_col_name:
    col_name [(length)] [ASC | DESC] # The length parameter is used only to create LSIs on
the shard keys of an index table.
index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
index_type:
    USING {BTREE | HASH}
algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}
lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

The `CREATE GLOBAL INDEX` statement is used to create a GSI for a table after the table is created. This statement introduces the GLOBAL keyword to the CREATE INDEX statement in MySQL. This keyword specifies that the type of the index to be created is GSI. Limits are imposed on creating a GSI for a table after the table is created. For more information about the limits on GSIs, see Notes for using GSIs.

For more information about the clauses that are used to define GSIs, see CREATE TABLE.

Examples

The following example shows how to create a common GSI for a table after the table is created.

- Create a GSI.

```
# Create a table.
CREATE TABLE t_order (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `order_id` varchar(20) DEFAULT NULL,
  `buyer_id` varchar(20) DEFAULT NULL,
  `seller_id` varchar(20) DEFAULT NULL,
  `order_snapshot` longtext DEFAULT NULL,
  `order_detail` longtext DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `l_i_order` (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
# Create a GSI.
ALTER TABLE t_order ADD UNIQUE GLOBAL INDEX `g_i_buyer` (`buyer_id`) COVERING (`order_sna
pshot`) dbpartition by hash(`buyer_id`);
```

  - Primary table: The data on the primary table t_order is partitioned into database shards but not further partitioned into table shards. The database uses hash sharding based on the order_id column.
  - Index table: The data on the index table g_i_buyer is partitioned into database shards but not further partitioned into table shards. The database uses hash sharding based on the buyer_id column. The order_snapshot column is specified as the covering column.
  - Clause used to define the GSI: `GLOBAL INDEX `g_i_seller` ON t_order (`seller_id`) dbpartiti on by hash(`seller_id`)` .

- The following sample code shows how to execute the `SHOW INDEX` statement to view the information about indexes that includes the LSI on the order_id shard key and GSIs on buyer_id, id, order_id, and order_snapshot. buyer_id is the shard key of the index table. id and order_id are the default covering columns. id is the primary key and order_id is the shard key of the primary table. order_snapshot is the covering column that is explicitly specified.

```
mysql> show index from t_order;
+---------+------------+-----------+--------------+----------------+-----------+---------
----+----------+--------+------+------------+----------+---------------+
| TABLE   | NON_UNIQUE | KEY_NAME  | SEQ_IN_INDEX | COLUMN_NAME    | COLLATION | CARDINAL
ITY | SUB_PART | PACKED | NULL | INDEX_TYPE | COMMENT  | INDEX_COMMENT |
+---------+------------+-----------+--------------+----------------+-----------+---------
----+----------+--------+------+------------+----------+---------------+
| t_order |          0 | PRIMARY   |            1 | id             | A         |
0 |     NULL | NULL   |      | BTREE      |          |               |
| t_order |          1 | l_i_order |            1 | order_id       | A         |
0 |     NULL | NULL   | YES  | BTREE      |          |               |
| t_order |          0 | g_i_buyer |            1 | buyer_id       | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | INDEX    |               |
| t_order |          1 | g_i_buyer |            2 | id             | NULL      |
0 |     NULL | NULL   |      | GLOBAL     | COVERING |               |
| t_order |          1 | g_i_buyer |            3 | order_id       | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
| t_order |          1 | g_i_buyer |            4 | order_snapshot | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
+---------+------------+-----------+--------------+----------------+-----------+---------
----+----------+--------+------+------------+----------+---------------+
```

- You can execute the `SHOW GLOBAL INDEX` statement to view only the GSI information. For more

information, see SHOW GLOBAL INDEX.

```
mysql> show global index from t_order;
+---------------------+---------+------------+-----------+-------------+-----------------
------------+------------+-----------------+---------------------+--------------------+
-----------------+--------------------+--------------------+--------+
| SCHEMA              | TABLE   | NON_UNIQUE | KEY_NAME  | INDEX_NAMES | COVERING_NAMES
| INDEX_TYPE | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COUNT | TB_PARTITION
_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT | STATUS |
+---------------------+---------+------------+-----------+-------------+-----------------
------------+------------+-----------------+---------------------+--------------------+
-----------------+--------------------+--------------------+--------+
| ZZY3_DRDS_LOCAL_APP | t_order | 0          | g_i_buyer | buyer_id    | id, order_id, or
der_snapshot | NULL       | buyer_id         | HASH               | 4                  |
| NULL                | NULL               | PUBLIC |
+---------------------+---------+------------+-----------+-------------+-----------------
------------+------------+-----------------+---------------------+--------------------+
-----------------+--------------------+--------------------+--------+
```

- The following sample code can be used to view the schema of the index table. The index table contains the primary key of the primary table, the database shard key and table shard key, the default covering columns, and the custom covering columns. The AUTO_INCREMENT attribute is removed from the primary key column. The LSI is removed from the primary table. By default, a local unique index is created on the index table that contains all the index columns of the GSI to achieve the global unique constraint of the primary table.

```
mysql> show create table g_i_buyer;
+-----------+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| Table     | Create Table
|
+-----------+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------+
| g_i_buyer | CREATE TABLE `g_i_buyer` (
  `id` bigint(11) NOT NULL,
  `order_id` varchar(20) DEFAULT NULL,
  `buyer_id` varchar(20) DEFAULT NULL,
  `order_snapshot` longtext,
  PRIMARY KEY (`id`),
  UNIQUE KEY `auto_shard_key_buyer_id` (`buyer_id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`buyer_id`) |
+-----------+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------+
```

4.7. DROP INDEX

This topic describes how to use the DROP INDEX statement to delete a local secondary index (LSI) and a global secondary index (GSI).

LSI

LSIs in Distributed Relational Database Service (DRDS) can be deleted by using the same method that is used to delete LSIs in MySQL databases. For more information, see DROP INDEX.

GSI

Syntax

```
# The index_name parameter is the name of the GSI that you want to delete.
DROP INDEX index_name ON tbl_name
```

4.8. CREATE VIEW

This topic describes how to use the CREATE VIEW statement to create a view for a PolarDB-X 1.0PolarDB-X instance.

Prerequisites

The version of the PolarDB-X 1.0 instance must be 5.4.5 or later.

Syntax

```
CREATE
    [OR REPLACE]
    VIEW view_name [(column_list)]
    AS select_statement
```

Examples

```
# Create a table.
CREATE TABLE t_order (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `order_id` varchar(20) DEFAULT NULL,
  `buyer_id` varchar(20) DEFAULT NULL,
  `seller_id` varchar(20) DEFAULT NULL,
  `order_snapshot` longtext DEFAULT NULL,
  `order_detail` longtext DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `l_i_order` (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
# Create a view.
create view t_detail as select order_id,order_detail from t_order;
# Query a view.
select * from t_detail;
```

4.9. DROP VIEW

This topic describes how to use the DROP VIEW statement to delete a view of a PolarDB-X 1.0 instance.

Prerequisites

The version of the PolarDB-X 1.0 instance must be 5.4.5 or later.

Syntax

```
DROP VIEW [IF EXISTS] view_name
```

Examples

```
# Create a view: create view v as select 1;
# Delete a view: drop view v;
```

4.10. DDL FAQ

This topic provides answers to commonly asked questions about the execution errors of data definition language (DDL) statements in PolarDB-X 1.0.

What can I do if an execution error occurs when I create a table?

A DDL statement is processed in distributed mode. An error may cause schema inconsistency among shards. Therefore, you must manually clean up the error. You can perform the following steps:

1. PolarDB-X 1.0 provides basic error description information, such as syntax errors. If the error message is too long, the system prompts you to run the SHOW WARNINGS command to view the execution failure cause for each database shard.
2. Run the SHOW TOPOLOGY command to view the topology of physical tables.

```
SHOW TOPOLOGY FROM multi_db_multi_tbl;
+------+-----------------+-----------------------+
| ID   | GROUP_NAME      | TABLE_NAME            |
+------+-----------------+-----------------------+
|    0 | corona_qatest_0 | multi_db_multi_tbl_00 |
|    1 | corona_qatest_0 | multi_db_multi_tbl_01 |
|    2 | corona_qatest_0 | multi_db_multi_tbl_02 |
|    3 | corona_qatest_1 | multi_db_multi_tbl_03 |
|    4 | corona_qatest_1 | multi_db_multi_tbl_04 |
|    5 | corona_qatest_1 | multi_db_multi_tbl_05 |
|    6 | corona_qatest_2 | multi_db_multi_tbl_06 |
|    7 | corona_qatest_2 | multi_db_multi_tbl_07 |
|    8 | corona_qatest_2 | multi_db_multi_tbl_08 |
|    9 | corona_qatest_3 | multi_db_multi_tbl_09 |
|   10 | corona_qatest_3 | multi_db_multi_tbl_10 |
|   11 | corona_qatest_3 | multi_db_multi_tbl_11 |
+------+-----------------+-----------------------+
12 rows in set (0.21 sec)
```

3. Run the `CHECK TABLE tablename` command to check whether the logical table has been created.

For example, the following example shows the scenario where a physical table shard of `multi_db_multi_tbl` failed to be created.

```
    mysql> check table multi_db_multi_tbl;
    +-----------------------------------------------+-------+----------+-------------
-------------------------------------------------------------+
    | TABLE                                         | OP    | MSG_TYPE | MSG_TEXT
|
    +-----------------------------------------------+-------+----------+-------------
-------------------------------------------------------------+
    | andor_mysql_qatest. multi_db_multi_tbl | check | Error    | Table 'corona_qatest_
0. multi_db_multi_tbl_02' doesn't exist |
    +-----------------------------------------------+-------+----------+-------------
-------------------------------------------------------------+
    1 row in set (0.16 sec)
```

4. Create or delete the table in idempotent mode to create or delete the remaining physical tables.

```
CREATE TABLE IF NOT EXISTS table1
(id int, name varchar(30), primary key(id))
dbpartition by hash(id);
DROP TABLE IF EXISTS table1;
```

What can I do if I failed to create an index or add a column?

The method for handling the failure when you create an index or add a column is similar to the preceding steps for the table creation failure. For more information, see Troubleshoot DDL exceptions.

5.DML

5.1. SELECT

This topic describes how to execute the SELECT statement to query data from one or more tables.

Syntax

```
SELECT
    [ALL | DISTINCT]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [FOR UPDATE]
```

Descriptions of the clauses in the SELECT statement:

- select_expr specifies the column to be queried. One SELECT statement must contain at least one select_expr expression.
- table_references specifies the tables from which data is retrieved.
- The WHERE clause specifies query criteria. If this clause is specified as where_condition, the system returns the rows that meet the requirement in where_condition. If this clause is not specified, all rows are returned.
- The GROUP BY clause supports the references to column names, expressions, and positions in output columns.
- The HAVING clause is similar to the WHERE clause. The difference is that the HAVING clause allows you to use aggregate functions.
- The ORDER BY clause specifies the order in which data is sorted. This clause supports the references to column names, expressions, and positions in output columns. You can also specify the sort direction, such as ASC (ascending order) and DESC (descending order).
- The OFFSET clause specifies the offset of an output result set. The LIMIT clause specifies the size of an output result set. The LIMIT clause allows you to specify one or two numeric parameters. The parameters must be integer constants. If you specify two parameters, the first parameter specifies the offset of the first row to be returned and the second parameter specifies the maximum number of rows to be returned. The initial offset of the first row is 0 instead of 1. To be compatible with PostgreSQL, MySQL also supports LIMIT and OFFSET.
- The FOR UPDATE clause applies an exclusive lock on each row of the query results. This prevents other transactions from concurrently updating the rows. This also prevents other transactions from concurrently reading the rows for which some transaction isolation levels are specified.

Note

- The expressions that are used in a WHERE clause cannot be used in a HAVING clause. For example, the following SQL statement 1 must be rewritten as SQL statement 2.

SQL statement 1:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

SQL statement 2:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- You can use aggregate functions in the HAVING clause but not in the WHERE clause.

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

- If the LIMIT clause contains two parameters, the first parameter indicates the offset of the first row that is returned, and the second parameter indicates the number of rows that are returned. If the LIMIT clause contains only one parameter, this parameter indicates the number of rows that are returned, and the default offset is 0.
- The GROUP BY clause does not support ASC or DESC.
- If both GROUP BY and ORDER BY are used, the expressions that follow ORDER BY must be included in a SELECT clause or a GROUP BY clause. For example, the following SQL statement is not supported:

```
SELECT user FROM users GROUP BY age ORDER BY salary;
```

- Aggregate functions and expressions that contain aggregate functions cannot be used in the ORDER BY clause. If you want to use such an expression, define the expression as a select_expr, assign an alias to the expression, and then reference the alias in the ORDER BY clause.
- Empty strings cannot be used as aliases.

JOIN

PolarDB-X 1.0 supports the following JOIN syntax in table_references of the SELECT statement:

```
table_references:
    escaped_table_reference [, escaped_table_reference] ...
escaped_table_reference:
    table_reference
  | { OJ table_reference }
table_reference:
    table_factor
  | join_table
table_factor:
    [schema_name.]tbl_name [[AS] alias] [index_hint_list]
  | table_subquery [AS] alias
  | ( table_references )
join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
join_condition:
    ON conditional_expr
  | USING (column_list)
index_hint_list:
    index_hint [, index_hint] ...
index_hint:
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | IGNORE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
  | FORCE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
index_list:
    index_name [, index_name] ...
```

To use the JOIN statements, consider the following factors:

- JOIN, CROSS JOIN, and INNER JOIN are syntactic equivalents. This is also the case in MySQL.
- An INNER JOIN statement without an ON clause is equivalent to using a comma (,). Both of them indicate a CROSS JOIN. For example, the following SQL statements are equivalent:

  ```
  SELECT * FROM t1 INNER JOIN t2 WHERE t1.id > 10
  SELECT * FROM t1, t2 WHERE t1.id > 10
  ```

- `USING(column_list)` is used to specify the column names that exist in both tables from which you want to combine data. PolarDB-X 1.0 constructs an equivalent condition based on these columns. For example, the following SQL fragments are equivalent:

  ```
  a LEFT JOIN b USING(c1, c2)
  a LEFT JOIN b ON a.c1 = b.c1 AND a.c2 = b.c2
  ```

- The JOIN operator has higher precedence than the comma operator (,). The JOIN expression t1, t2 JOIN t3 is interpreted as (t1, (t2 JOIN t3)), not as ((t1, t2) JOIN t3).
- LEFT JOIN and RIGHT JOIN must contain the ON condition.
- index_hint specifies the index to be used by MySQL. PolarDB-X 1.0 pushes the hint to the underlying MySQL database.

- STRAIGHT_JOIN and NATURAL JOIN are not supported.

UNION

PolarDB-X 1.0 supports the following UNION syntax:

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

> **Note** In each SELECT clause of UNION, PolarDB-X 1.0 does not support multiple columns with the same name. The following SQL statement is not supported because the column names in the SELECT clause are duplicates.
>
> ```
> SELECT id, id, name FROM t1 UNION SELECT pk, pk, name FROM t2;
> ```

References

- SELECT syntax in MySQL
- JOIN syntax in MySQL
- UNION syntax in MySQL

5.2. Subquery

This topic describes the types of subqueries supported by PolarDB-X 1.0 and the limits and additional considerations when you use subqueries in PolarDB-X 1.0.

Limits

Compared with the native MySQL, PolarDB-X 1.0 has the following limits when you use subqueries:

- Subqueries cannot be used in HAVING clauses. Example:

  ```
  SELECT name, AVG( quantity )
  FROM tb1
  GROUP BY name
  HAVING AVG( quantity ) > 2* (
     SELECT AVG( quantity )
     FROM tb2
   );
  ```

- Subqueries cannot be used in JOIN ON clauses. Example:

  ```
  SELECT * FROM tb1 p JOIN tb2 s on (p.id=s.id and p.quantity>All(select quantity from tb3)
  )
  ```

- ROW subqueries and scalar subqueries cannot be placed before and after equal signs (=) simultaneously. Example:

  ```
  select * from tb1 where row(id, name) = (select id, name from tb2)
  ```

- Subqueries cannot be used in UPDATE SET clauses. Example:

```
UPDATE t1 SET c1 = (SELECT c2 FROM t2 WHERE t1.c1 = t2.c1) LIMIT 10
```

Additional considerations

In PolarDB-X 1.0, some subqueries can be executed by using only the APPLY operator and result in inefficient queries. Avoid the following inefficient SQL statements:

- SQL statements whose WHERE clauses contain both OR operators and subqueries. The execution efficiency is reduced based on the data in the foreign tables. Examples:

```
Efficient: select * from tb1 where id in (select id from tb2)
Efficient: select * from tb1 where id in (select id from tb2) and id>3
Inefficient: select * from tb1 where id in (select id from tb2) or  id>3
```

- Correlated subqueries whose correlated items are used in functions or used along with non-equal signs. Examples:

```
Efficient: select * from tb1 a where id in
      (select id from tb2 b where a.name=b.name)
Inefficient: select * from tb1 a where id in
      (select id from tb2 b where UPPER(a.name)=b.name)
Inefficient: select * from tb1 a where id in
      (select id from tb2 b where a.decimal_test=abs(b.decimal_test))
Inefficient: select * from tb1 a where id in
      (select id from tb2 b where a.name! =b.name)
Inefficient: select * from tb1 a where id in
      (select id from tb2 b where a.name>=b.name)
```

- Correlated subqueries whose correlated items are connected with other conditions by using OR operators. Examples:

```
Efficient: select * from tb1 a where id in
      (select id from tb2 b where a.name=b.name
                                and b.date_test<'2015-12-02')
Inefficient: select * from tb1 a where id in
      (select id from tb2 b where a.name=b.name
                                or b.date_test<'2015-12-02')
Inefficient: select * from tb1 a where id in
      (select id from tb2 b where a.name=b.name
                                or b.date_test=a.date_test)
```

- Scalar subqueries that have correlated items. Examples:

```
Efficient: select * from tb1 a where id >
        (select id from tb2 b where b.date_test<'2015-12-02')
Inefficient: select * from tb1 a where id >
        (select id from tb2 b where a.name=b.name
                                  and b.date_test<'2015-12-02')
```

- Subqueries whose correlated items span the correlation levels. Examples:

- An SQL statement has multiple correlation levels. The correlated items in each subquery are correlated only with the upper level. Such statements are efficient.

```
Efficient: select * from tb1 a where id in(select id from tb2 b
        where a.name=b.name and
        exists (select name from tb3 c where b.address=c.address))
```

- An SQL statement has multiple correlation levels. The correlated items of subqueries in `table c` are correlated with columns in `table a`. Such statements are inefficient.

```
Inefficient: select * from tb1 a where id in(select id from tb2 b
        where a.name=b.name and
        exists (select name from tb3 c where a.address=c.address))
```

> **Note** In the preceding example, both `table a` and `table b`, `table b` and `table c` belong to the same correlation level. The correlation between `table a` and `table c` spans the correlation levels.

- Subqueries that contain GROUP BY clauses. Make sure that the correlated items are correlated to the grouping columns. Examples:
  - An SQL subquery contains aggregate functions and correlated items. The `b.pk` correlated item is correlated to the `pk` grouping column. Such SQL statements are efficient.

```
Efficient: select * from tb1 a where exists
    (select pk from tb2 b
                where a.pk=b.pk and  b.date_test='2003-04-05'
                group by pk);
```

  - An SQL subquery contains aggregate functions and correlated items. The `b.date_test` correlated item is not correlated to the `pk` grouping column. Such SQL statements are inefficient.

```
Inefficient: select * from tb1 a where exists
    (select pk from tb2 b
                where a.date_test=b.date_test and b.date_test='2003-04-05'
                group by pk);
```

Supported subqueries

PolarDB-X 1.0 supports the following types of subqueries:

- Comparisons using subqueries

  Comparisons using subqueries indicate subqueries that use comparison operators. These subqueries are commonly used.

  - Syntax:

```
non_subquery_operand comparison_operator (subquery)
comparison_operator: = > < >= <= <> ! =  <=> like
```

- Example:

```
select * from tb1 WHERE 'a' = (SELECT column1 FROM t1)
```

> **Note** Subqueries can be placed only to the right of comparison operators.

- Subqueries with ANY, ALL, IN/NOT IN, and EXISTS/NOT EXISTS
  - Syntax:

```
operand comparison_operator ANY (subquery)
operand comparison_operator ALL (subquery)
operand IN (subquery)
operand NOT IN (subquery)
operand EXISTS (subquery)
operand NOT EXISTS (subquery)
comparison_operator:=  >  <  >=  <=  <>  ! =
```

  - Examples
    - ANY: If any row returned by the subquery meets the expression before ANY, TRUE is returned. Otherwise, FALSE is returned.
    - ALL: If all rows returned by the subquery meet the expression before ALL, TRUE is returned. Otherwise, FALSE is returned.
    - IN: If IN is used before the subquery, IN is equivalent to `=ANY`. Example:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

    - NOT IN: If NOT IN is used before the subquery, NOT IN is equivalent to `<>ALL`. Example:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

    - EXISTS: If the subquery returns any rows, TRUE is returned. Otherwise, FALSE is returned. Example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

> **Note** If a subquery contains any rows, the WHERE condition returns TRUE even if the subquery contains only NULL rows.

    - NOT EXISTS: If the subquery returns any rows, FALSE is returned. Otherwise, TRUE is returned.

- ROW subqueries
  - ROW subqueries support the following comparison operators:

```
comparison_operator: =  >  <  >=  <=  <>  ! =  <=>
```

- Examples:

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

The preceding two SQL statements are equivalent. Data rows in table t1 are returned only when the following conditions are met:

- The subquery (`SELECT col3, col4 FROM t2 WHERE id=10`) returns only one row. An error is reported if multiple rows are returned.
- `col3` and `col4` returned by the subquery are equal to `col1` and `col2` in the primary table.

- Correlated subqueries

  Correlated subqueries are subqueries that contain references to foreign tables in outer queries. Example:

```
SELECT * FROM t1
  WHERE column1 = ANY (SELECT column1 FROM t2
                       WHERE t2.column2 = t1.column2);
```

  In the example, the subquery does not contain table t1 and its column column2. In this case, the subquery finds the table in the outer query.

- Derived tables (subqueries in a FROM clause)

  Derived tables are subqueries in a FROM clause.

  - Syntax:

```
SELECT ... FROM (subquery) [AS] tbl_name ...
```

- Examples
  a. Prepare data:

     Execute the following statements to create table t1:

     ```
     CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
     INSERT INTO t1 VALUES (1,'1',1.0);
     INSERT INTO t1 VALUES (2,'2',2.0);
     ```

     Execute the following statement. The query result is `2, '2', 4.0`.

     ```
     SELECT sb1,sb2,sb3
       FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
       WHERE sb1 > 1;
     ```

  b. Query data: Query the average value of grouped data that is processed by the SUM function.

     If you execute the following SQL statement, an error is reported and no result is returned.

     ```
     SELECT AVG(SUM(s1)) FROM t1 GROUP BY s1;
     ```

     You can execute the following statement that contains a derived table. The query result is `1.5000`.

     ```
     SELECT AVG(sum_s1)
       FROM (SELECT SUM(s1) AS sum_s1
             FROM t1 GROUP BY s1) AS t1;
     ```

     > **Note**
     > - A derived table must have an alias, such as `t1` in the previous statement.
     > - A derived table can return a scalar, a column, a row, or a table.
     > - Derived tables cannot be correlated subqueries. Derived tables cannot contain references to foreign tables in outer queries.

5.3. INSERT

You can execute the INSERT statements to insert data into tables.

Syntax

the primary key.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] [schema_name.]tbl_name
[(col_name [, col_name] ...)]
{VALUES | VALUE} (value_list) [, (value_list)]
[ON DUPLICATE KEY UPDATE assignment_list]
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] [schema_name.]tbl_name
SET assignment_list
[ON DUPLICATE KEY UPDATE assignment_list]
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] [schema_name.]tbl_name
[(col_name [, col_name] ...)]
SELECT ...
[ON DUPLICATE KEY UPDATE assignment_list]
value_list:
value [, value] ...
value:
{expr | DEFAULT}
assignment_list:
assignment [, assignment] ...
assignment:
col_name = value
```

Limits on syntax

The following INSERT statements are not supported:

- INSERT IGNORE ON DUPLICATE KEY UPDATE.

```
INSERT IGNORE INTO tb (id) VALUES(7) ON DUPLICATE KEY UPDATE id = id + 1;
```

- INSERT statements that contain PARTITION functions.

```
INSERT INTO tb PARTITION (p0) (id) VALUES(7);
```

- INSERT statements where the NEXTVAL functions are nested.

```
INSERT INTO tb(id) VALUES(SEQ1.NEXTVAL + 1);
```

- INSERT statements that contain column names.

```
INSERT INTO tb(id1, id2) VALUES(1, id1 + 1);
```

Limits on distributed transactions

> **Note** If a transaction is processed in the same database shard even when you use table shards, this transaction is considered as a single-database transaction. For example, a transaction contains the shard key and the INSERT or UPDATE statement in the transaction is executed in the same database shard. In this case, this transaction is a single-database transaction.

If the distributed transaction feature is enabled, the INSERT statements that meet the following conditions are not supported:

- No primary key is specified for the table to which data is to be inserted. The following statements are

used as examples:

```
CREATE TABLE tb(id INT, name VARCHAR(10));
INSERT INTO tb VALUES(1, 'a');
```

- The table to which data is to be inserted is not sharded. The primary key values are auto-incremented, but no Distributed Relational Database Service (DRDS) sequence is used for the primary key. For more information about DRDS sequences. The following statements are used as examples:

```
CREATE TABLE tb(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(10));
INSERT INTO tb(name) VALUES('a');    # The statements are not supported.
```

You can specify a DRDS sequence for the primary key to enable the preceding statements to be supported. For more information about DRDS sequences. The following statements are used as examples:

```
CREATE TABLE tb(id INT PRIMARY KEY AUTO_INCREMENT BY GROUP, name VARCHAR(10));
INSERT INTO tb(name) VALUES('a');    # The statements are supported.
```

References

INSERT Statement for the native MySQL.

5.4. REPLACE

You can use the REPLACE syntax to insert rows to tables or replace rows in tables.

Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] [schema_name.]tbl_name
[(col_name [, col_name] ...)]
{VALUES | VALUE} (value_list) [, (value_list)]
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] [schema_name.]tbl_name
SET assignment_list
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] [schema_name.]tbl_name
[(col_name [, col_name] ...)]
SELECT ...
value_list:
value [, value] ...
value:
{expr | DEFAULT}
assignment_list:
assignment [, assignment] ...
assignment:
col_name = value
```

Limits on syntax

The following syntax is not supported:

- Syntax that contains PARTITION. The following example shows the syntax:

```
REPLACE INTO tb PARTITION (p0) (id) VALUES(7);
```

- Syntax where NEXTVAL is nested. The following example shows the syntax:

```
REPLACE INTO tb(id) VALUES(SEQ1.NEXTVAL + 1);
```

- Syntax that contains column names. The following example shows the syntax:

```
REPLACE INTO tb(id1, id2) VALUES(1, id1 + 1);
```

Limits on distributed transactions

> **Note** If you use table shards, but a transaction is processed in the same database (for example, INSERT or UPDATE contains the shard key), this transaction is considered as a single-database transaction.

When the distributed transaction feature is enabled, the following REPLACE command is not supported:

- No primary key is specified for the table, as shown in the following example:

```
CREATE TABLE tb(id INT, name VARCHAR(10));
REPLACE INTO tb VALUES(1, 'a');
```

- The table is not sharded. The primary key is auto-incremented, but no sequence is used for the primary key. For more information about sequences. The following example shows the corresponding statements:

```
CREATE TABLE tb(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(10));
REPLACE INTO tb(name) VALUES('a');
```

  You can specify a sequence for the primary key to prevent the limit. For more information about sequences. The following example shows the corresponding statements:

```
CREATE TABLE tb(id INT PRIMARY KEY AUTO_INCREMENT BY GROUP, name VARCHAR(10));
REPLACE INTO tb(name) VALUES('a');
```

References

REPLACE syntax for MySQL

5.5. UPDATE

You can use the UPDATE syntax to modify the rows that meet the conditions in tables.

Syntax

- Single logical table.

```
UPDATE [LOW_PRIORITY] [IGNORE] [schema_name.]tbl_name
    SET assignment_list
    [WHERE where_condition]
value:
    {expr | DEFAULT}
assignment:
    col_name = value
assignment_list:
    assignment [, assignment] ...
```

- Multiple logical tables.

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET assignment_list
    [WHERE where_condition]
```

> **Note**
> - The UPDATE statements support the following modifiers:
>   - If you specify LOW_PRIORITY, the UPDATE operation is performed after all the read operations on the table are completed.
>   - If you specify IGNORE, the errors are ignored during the update process. This indicates that the update is not interrupted by the errors.
> - Each modifier in the UPDATE statements is pushed down to the storage layer MySQL and remains unchanged. This process does not affect the modifier operations of PolarDB-X 1.0.

Limits on syntax

Compared with the UPDATE syntax of native MySQL, the UPDATE syntax of PolarDB-X 1.0 has the following limits:

- Correlated and uncorrelated subqueries are not supported in the SET clauses. This limit is illustrated in the following example:

```
UPDATE t1 SET name = (SELECT name FROM t2 WHERE t2.id = t1.id) WHERE id > 10;
```

- By default, an UPDATE statement is forbidden if the statement needs to update more than 10,000 rows and the statement cannot be pushed down. In this case, you must use hints so that the UPDATE statement can be supported, as shown in the following example:

```
UPDATE t1 SET t1.name = "abc" ORDER BY name LIMIT 10001;
UPDATE t1, t2 SET t1.name = t2.name WHERE t1.id = t2.name LIMIT 10001;
```

> **Note** The shard key of t1 and t2 is ID.

References

UPDATE syntax for MySQL.

5.6. DELETE

You can execute the DELETE statements to delete the rows that meet the conditions from tables.

Syntax

The following DELETE statements delete the rows that meet the conditions specified by `where_condition` from the tables specified by `tbl_name`, and return the number of deleted rows. If you do not specify the WHERE conditions, all the data in the specified tables is deleted.

- Single logical table.

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM [schema_name.]tbl_name
    [WHERE where_condition]
```

- Multiple logical tables.

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[. *]] ...
    FROM table_references
    [WHERE where_condition]
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM [schema_name.]tbl_name[.*] [, [schema_name.]tbl_name[. *]] ...
    USING table_references
    [WHERE where_condition]
```

> **Note**
>
> - The DELETE statements support the following modifiers:
>   - If you specify LOW_PRIORITY, the DELETE operation is performed after all the read operations from the table are completed.
>   - If you specify IGNORE, the errors are ignored during the deletion process.
>   - QUICK is related to the storage engines of MySQL. For more information, see MySQL documentation.
> - Each modifier in the DELETE statements is pushed down to the storage layer MySQL and remains unchanged. This process does not affect the modifier operations of PolarDB-X 1.0.

Limits on syntax

Compared with the DELETE syntax of the native MySQL, the DELETE syntax of PolarDB-X 1.0 has the following limits:

By default, a DELETE statement is forbidden if the statement needs to delete more than 10,000 rows and the statement cannot be pushed down. In this case, you must use hints so that DELETE statement can be supported, as shown in the following example:

```
DELETE FROM t1 ORDER BY name LIMIT 10001;
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3 WHERE t1.id=t2.id AND t2.id=t3.name LIMIT
10001;
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3 WHERE t1.id=t2.id AND t2.id=t3.name
LIMIT 10001;
```

> **Note** The shard key of t1, t2, and t3 is ID.

5.7. Limits of global secondary indexes on DML

This topic describes the limits that global secondary indexes (GSIs) in PolarDB-X 1.0 have on data manipulation language (DML).

Prerequisites

The versions of the custom ApsaraDB RDS for MySQL instances are 5.7 or later, and the versions of the PolarDB-X 1.0 instances are 5.4.1 or later.

Examples

The following table is used to describe the limits that GSIs have on DML.

```
CREATE TABLE t_order(
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `order_id` varchar(20) DEFAULT NULL,
  `buyer_id` varchar(20) DEFAULT NULL,
  `seller_id` varchar(20) DEFAULT NULL,
  `order_snapshot` longtext DEFAULT NULL,
  `order_detail` longtext DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `l_i_order` (`order_id`),
  GLOBAL INDEX `g_i_seller` (`seller_id`) dbpartition by hash(`seller_id`) tbpartition by h
ash(`seller_id`),
  GLOBAL UNIQUE INDEX `g_i_buyer` (`buyer_id`) COVERING (order_snapshot) dbpartition by has
h(`buyer_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
```

After a GSI fails to be written to a table, other DML statements cannot be executed on the table and transactions cannot be committed on the table.

```
SET DRDS_TRANSACTION_POLICY='XA';
INSERT INTO t_order(order_id, buyer_id, seller_id) VALUES('order_1', 'buyer_1', 'seller_1')
;
# A GSI failed to be written to the table.
INSERT IGNORE INTO t_order(order_id, buyer_id, seller_id) VALUES('order_2', 'buyer_1', 'sel
ler_1');
# Other DML statements cannot be executed on the table.
INSERT IGNORE INTO t_order(order_id, buyer_id, seller_id) VALUES('order_2', 'buyer_2', 'sel
ler_2');
# Transactions cannot be committed on the table.
COMMIT;
```

6.SHOW

6.1. SHOW HELP

This topic describes how to view all auxiliary SQL commands in Distributed Relational Database Service (DRDS) by executing the SHOW HELP statement.

Context

```
mysql> show help;
+---------------------------------------+-----------------------------------------------
---------+-----------------------------------------------+
| STATEMENT                             | DESCRIPTION
| EXAMPLE                                  |
+---------------------------------------+-----------------------------------------------
---------+-----------------------------------------------+
| show rule                             | Report all table rule
|                                          |
| show rule from TABLE                  | Report table rule
| show rule from user                      |
| show full rule from TABLE             | Report table full rule
| show full rule from user                 |
| show topology from TABLE              | Report table physical topology
| show topology from user                  |
| show partitions from TABLE            | Report table dbPartition or tbPartition columns
| show partitions from user                |
| show broadcasts                       | Report all broadcast tables
|                                          |
| show datasources                      | Report all partition db threadPool info
|                                          |
| show node                             | Report master/slave read status
|                                          |
| show slow                             | Report top 100 slow sql
|                                          |
| show physical_slow                    | Report top 100 physical slow sql
|                                          |
| clear slow                            | Clear slow data
|                                          |
| trace SQL                             | Start trace sql, use show trace to print profil
ing data | trace select count(*) from user; show trace |
| show trace                            | Report sql execute profiling info
|                                          |
| explain SQL                           | Report sql plan info
| explain select count(*) from user        |
| explain detail SQL                    | Report sql detail plan info
| explain detail select count(*) from user |
| explain execute SQL                   | Report sql on physical db plan info
| explain execute select count(*) from user |
| show sequences                        | Report all sequences status
|                                          |
| create sequence NAME [start with COUNT] | Create sequence
| create sequence test start with 0        |
| alter sequence NAME [start with COUNT]  | Alter sequence
| alter sequence test start with 100000    |
| drop sequence NAME                    | Drop sequence
| drop sequence test                       |
+---------------------------------------+-----------------------------------------------
---------+-----------------------------------------------+
20 rows in set (0.00 sec)
```

6.2. Rule and topology query statements

This topic describes rule and topology query statements.

SHOW RULE [FROM tablename]

Description:

- `SHOW RULE` : queries the sharding details of each logical table in a database.
- `SHOW RULE FROM tablename` : queries the sharding details of a specified logical table in a database.

```
mysql> show rule;
+------+--------------+-----------+-----------------+---------------------+---------------
-----+------------------+---------------------+--------------------+
| ID   | TABLE_NAME   | BROADCAST | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_C
OUNT | TB_PARTITION_KEY | TB_PARTITION_POLICY | TB_PARTITION_COUNT |
+------+--------------+-----------+-----------------+---------------------+---------------
-----+------------------+---------------------+--------------------+
|    0 | dept_manager |         0 |                 | NULL                | 1
|                  | NULL                |                   1 |                    |
|    1 | emp          |         0 | emp_no          | hash                | 8
| id               | hash                |                   2 |                    |
|    2 | example      |         0 | shard_key       | hash                | 8
|                  | NULL                |                   1 |                    |
+------+--------------+-----------+-----------------+---------------------+---------------
-----+------------------+---------------------+--------------------+
3 rows in set (0.01 sec)
```

Important columns:

- **BROADCAST** : indicates whether the table is a broadcast table. A value of 0 indicates that the table is not a broadcast table. A value of 1 indicates that the table is a broadcast table.
- **DB_PARTITION_KEY**: indicates the database shard key. If no database shards exist, the parameter value is empty.
- **DB_PARTITION_POLICY**: indicates the database sharding policy. The parameter values can be hash values and date values in the formats such as YYYYMM, YYYYDD, and YYYYWEEK.
- **DB_PARTITION_COUNT** : indicates the number of database shards.
- **TB_PARTITION_KEY**: indicates the table shard key. If no table shards exist, the parameter value is empty.
- **TB_PARTITION_POLICY**: indicates the table sharding policy. The parameter values can be hash

values or date values in the formats such as MM, DD, MMDD, and WEEK.

- **TB_PARTITION_COUNT**: indicates the number of table shards.

SHOW FULL RULE [FROM tablename]

You can execute this SQL statement to view the sharding rules of the logical tables in a database. This statement queries more detailed information than the SHOW RULE statement.

```
mysql> show full rule;
+------+--------------+-----------+------------+----------------------+------------------
---------------------------------+-----------------------------------------------------------+---
-------------+---------------------------------------+----------------+---------------------
---------------------------+
| ID   | TABLE_NAME   | BROADCAST | JOIN_GROUP | ALLOW_FULL_TABLE_SCAN | DB_NAME_PATTERN
| DB_RULES_STR                                        | TB_NAME_PATTERN | TB_RULES_STR
| PARTITION_KEYS | DEFAULT_DB_INDEX                                |
+------+--------------+-----------+------------+----------------------+------------------
---------------------------------+-----------------------------------------------------------+---
-------------+---------------------------------------+----------------+---------------------
---------------------------+
|    0 | dept_manager |         0 | NULL       |                    0 | SEQ_TEST_148776778
0814RGKKSEQ_TEST_WNJG_0000_RDS   | NULL                                                      | de
pt_manager    | NULL                                | NULL           | SEQ_TEST_148776778081
4RGKKSEQ_TEST_WNJG_0000_RDS |
|    1 | emp          |         0 | NULL       |                    1 | SEQ_TEST_148776778
0814RGKKSEQ_TEST_WNJG_{0000}_RDS | ((#emp_no,1,8#).longValue().abs() % 8)              | em
p_{0}         | ((#id,1,2#).longValue().abs() % 2) | emp_no      id | SEQ_TEST_148776778081
4RGKKSEQ_TEST_WNJG_0000_RDS |
|    2 | example      |         0 | NULL       |                    1 | SEQ_TEST_148776778
0814RGKKSEQ_TEST_WNJG_{0000}_RDS | ((#shard_key,1,8#).longValue().abs() % 8).intdiv(1) | ex
ample         | NULL                                | shard_key      | SEQ_TEST_148776778081
4RGKKSEQ_TEST_WNJG_0000_RDS |
+------+--------------+-----------+------------+----------------------+------------------
---------------------------------+-----------------------------------------------------------+---
-------------+---------------------------------------+----------------+---------------------
---------------------------+
3 rows in set (0.01 sec)
```

Important columns:

- **BROADCAST**: indicates whether the table is a broadcast table. A value of 0 indicates that the table is not a broadcast table. A value of 1 indicates that the table is a broadcast table.
- **JOIN_GROUP**: indicates a reserved field.
- **ALLOW_FULL_TABLE_SCAN**: indicates whether data querying is allowed if no table shard keys are specified for sharding. If this parameter is set to true, each physical table is scanned to locate the data that meets the condition. This is a full table scan.
- **DB_NAME_PATTERN**: The digit 0 inside a pair of braces {} in the parameter value is a placeholder. When the SQL statement is executed, the placeholders are replaced by the value of DB_RULES_STR. The number of digits in the parameter value remains unchanged. For example, if the value of DB_NAME_PATTERN is SEQ_{0000}_RDS and the value of DB_RULES_STR is [1,2,3,4], the following DB_NAME values are generated: SEQ_0001_RDS, SEQ_0002_RDS, SEQ_0003_RDS, and SEQ_0004_RDS.

- **DB_RULES_STR**: indicates the database sharding rule.
- **TB_NAME_PATTERN**: The digit 0 inside a pair of braces {} in the parameter value is a placeholder. When the SQL statement is executed, the placeholders are replaced by the value of TB_RULES_STR. The number of digits in the parameter value remains unchanged. For example, if the value of TB_NAME_PATTERN is table_{00} and the value of TB_RULES_STR is [1,2,3,4,5,6,7,8], the following tables are generated: table_01, table_02, table_03, table_04, table_05, table_06, table_07, and table_08.
- **TB_RULES_STR**: indicates the table sharding rule.
- **PARTITION_KEYS**: indicates a set of the database and table shard keys. If both database sharding and table sharding are performed, the database shard key is placed before the table shard key.
- **DEFAULT_DB_INDEX**: indicates the database shard in which a single-database non-partitioned table is stored.

SHOW TOPOLOGY FROM tablename

You can execute this SQL statement to view the topology of a specified logical table. The information contains the database shards to which data in the logical table is partitioned and the table shards in each database shard.

```
mysql> show topology from emp;
+------+-------------------------------------------------+------------+
| ID   | GROUP_NAME                                      | TABLE_NAME |
+------+-------------------------------------------------+------------+
|    0 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0000_RDS | emp_0      |
|    1 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0000_RDS | emp_1      |
|    2 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0001_RDS | emp_0      |
|    3 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0001_RDS | emp_1      |
|    4 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0002_RDS | emp_0      |
|    5 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0002_RDS | emp_1      |
|    6 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0003_RDS | emp_0      |
|    7 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0003_RDS | emp_1      |
|    8 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0004_RDS | emp_0      |
|    9 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0004_RDS | emp_1      |
|   10 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0005_RDS | emp_0      |
|   11 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0005_RDS | emp_1      |
|   12 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0006_RDS | emp_0      |
|   13 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0006_RDS | emp_1      |
|   14 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0007_RDS | emp_0      |
|   15 | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0007_RDS | emp_1      |
+------+-------------------------------------------------+------------+
16 rows in set (0.01 sec)
```

SHOW PARTITIONS FROM tablename

You can execute this SQL statement to view a set of database and table shard keys, which are separated by commas (,). If two values are returned, both database sharding and table sharding are performed. The first value is the database shard key and the second value is the table shard key. If only one value is returned, only database sharding is performed. This value is the database shard key.

```
mysql> show partitions from emp;
+-----------+
| KEYS      |
+-----------+
| emp_no,id |
+-----------+
1 row in set (0.00 sec)
```

SHOW BROADCASTS

You can execute this SQL statement to view the broadcast tables.

```
mysql> show broadcasts;
+------+------------+
| ID   | TABLE_NAME |
+------+------------+
|    0 | brd2       |
|    1 | brd_tbl    |
+------+------------+
2 rows in set (0.01 sec)
```

SHOW DATASOURCES

You can execute this SQL statement to view the information about the underlying storage. The information includes the database name, database group name, connection URL, username, storage type, read and write weights, and connection pool information.

```
mysql> show datasources;
+------+----------------------------+------------------------------------------------+-----
---------------------------------------------+--------------------------------------------
-------------------------------------+-----------+-------+------+------+------+-----------
--+----------+--------------+---------------+---------------------------------------------
+-------------+--------------+
| ID   | SCHEMA                     | NAME                                           | GROU
P                                            | URL
| USER      | TYPE  | INIT | MIN  | MAX  | IDLE_TIMEOUT | MAX_WAIT | ACTIVE_COUNT | POOLING
_COUNT | ATOM                                         | READ_WEIGHT | WRITE_WEIGHT |
+------+----------------------------+------------------------------------------------+-----
---------------------------------------------+--------------------------------------------
-------------------------------------+-----------+-------+------+------+------+-----------
--+----------+--------------+---------------+---------------------------------------------
+-------------+--------------+
|    0 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0000_iiab_1 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0000_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0000 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0000_iiab |
10          | 10           |
|    1 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0001_iiab_2 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0001_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0001 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0001_iiab |
```

```
10          | 10           |
|    2 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0002_iiab_3 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0002_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0002 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0002_iiab |
10          | 10           |
|    3 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0003_iiab_4 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0003_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0003 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0003_iiab |
10          | 10           |
|    4 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0004_iiab_5 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0004_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0004 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0004_iiab |
10          | 10           |
|    5 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0005_iiab_6 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0005_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0005 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0005_iiab |
10          | 10           |
|    6 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0006_iiab_7 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0006_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0006 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0006_iiab |
10          | 10           |
|    7 | seq_test_1487767780814rgkk | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0007_iiab_8 | SEQ_
TEST_1487767780814RGKKSEQ_TEST_WNJG_0007_RDS | jdbc:mysql://rds1ur80kcv8g3t6p3ol.mysql.rds.
aliyuncs.com:3306/seq_test_wnjg_0007 | jnkinsea0 | mysql | 0    | 24   | 72   | 15
| 5000     | 0            | 1             | rds1ur80kcv8g3t6p3ol_seq_test_wnjg_0007_iiab |
10          | 10           |
+------+----------------------------+-----------------------------------------------+-----
-------------------------------------------+-----------------------------------------------
-----------------------------------+-----------+-------+------+------+------+------------
--+----------+--------------+---------------+-----------------------------------------------
+-------------+--------------+
8 rows in set (0.01 sec)
```

Important columns:

- **SCHEMA**: indicates the database name.
- **GROUP**: indicates the database group name. After the databases are grouped, you can manage multiple databases that store the same data in a group. For example, after you replicate the data of a database to an ApsaraDB RDS for MySQL instance, you can manage the primary database and the secondary database in a group. Database grouping enables read/write splitting and primary/secondary switchovers.
- **URL**: indicates the URL that is used to connect to an underlying ApsaraDB RDS for MySQL database.
- **TYPE**: indicates the underlying storage type. Only ApsaraDB RDS for MySQL is supported.
- **READ_WEIGHT**: indicates the read weight. If you want to reduce the number of read requests to the primary ApsaraDB RDS for MySQL instance, you can use the read/write splitting feature to distribute some read requests to the secondary ApsaraDB RDS for MySQL instances. This offloads the read

requests from the primary ApsaraDB RDS for MySQL instance. PolarDB-X 1.0 automatically identifies the read and write requests. Then, it sends the write requests to the primary ApsaraDB RDS for MySQL instance and distributes the read requests to each ApsaraDB RDS for MySQL instance based on the specified read weights.

- **WRITE_WEIGHT**: indicates the write weight.

SHOW NODE

You can execute this SQL statement to view the data of a physical database, such as the accumulative number of read operations, the accumulative number of write operations, the accumulative read weights, and the accumulative write weights.

```
mysql> show node;
+------+-----------------------------------------------------+-------------------+------------
------+---------------------+--------------------+
| ID   | NAME                                                | MASTER_READ_COUNT | SLAVE_READ_
COUNT | MASTER_READ_PERCENT | SLAVE_READ_PERCENT |
+------+-----------------------------------------------------+-------------------+------------
------+---------------------+--------------------+
| 0    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0000_RDS |                12 |
0 | 100%                | 0%                 |
| 1    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0001_RDS |                 0 |
0 | 0%                  | 0%                 |
| 2    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0002_RDS |                 0 |
0 | 0%                  | 0%                 |
| 3    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0003_RDS |                 0 |
0 | 0%                  | 0%                 |
| 4    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0004_RDS |                 0 |
0 | 0%                  | 0%                 |
| 5    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0005_RDS |                 0 |
0 | 0%                  | 0%                 |
| 6    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0006_RDS |                 0 |
0 | 0%                  | 0%                 |
| 7    | SEQ_TEST_1487767780814RGKKSEQ_TEST_WNJG_0007_RDS |                 0 |
0 | 0%                  | 0%                 |
+------+-----------------------------------------------------+-------------------+------------
------+---------------------+--------------------+
8 rows in set (0.01 sec)
```

Important columns:

- **MASTER_COUNT**: indicates the accumulative number of read and write queries processed by the primary ApsaraDB RDS for MySQL instance.
- **SLAVE_COUNT**: indicates the accumulative number of read-only queries processed by the secondary ApsaraDB RDS for MySQL instances.
- **MASTER_PERCENT**: indicates the actual percentage of the accumulative read and write queries processed by the primary ApsaraDB RDS for MySQL instance. This is not the specified percentage.
- **SLAVE_PERCENT**: indicates the actual percentage of the accumulative read and write queries processed by the secondary ApsaraDB RDS for MySQL instances. This is not the specified percentage.

> **Note**
> - Read-only queries in transactions are sent to the primary ApsaraDB RDS for MySQL instance.
> - The `MASTER_PERCENT` and `SLAVE_PERCENT` columns indicate the accumulative historical data. If the ratio between the read weight and the write weight changes, these parameter values do not immediately reflect the latest ratio. The latest ratio appears after a long period of time.

6.3. Slow SQL queries

This topic describes how to execute the SHOW statements to identify slow SQL queries.

- SHOW [FULL] SLOW [WHERE expr] [limit expr]
- SHOW [FULL] PHYSICAL_SLOW [WHERE expr] [limit expr]
- CLEAR SLOW

SHOW [FULL] SLOW [WHERE expr] [limit expr]

SQL queries that take more than 1s are slow SQL queries.Logical slow SQL queries are sent from an application to a PolarDB-X 1.0 instance.

- `SHOW SLOW` : queries the top 100 logical slow SQL queries since a PolarDB-X 1.0 instance is started or the last `CLEAR SLOW` statement is executed.

  > **Note** The SHOW SLOW statement returns top 100 logical slow SQL queries. The returned data is stored in PolarDB-X 1.0. If you restart the database instance or execute the `CLEAR SLOW` statement, the returned data is cleared.

- `SHOW FULL SLOW` : queries all the logical slow SQL queries that are persistently stored in PolarDB-X 1.0 since the database instance is started. The system can retain a limited number of slow queries. The upper limit varies based on the instance type. PolarDB-X 1.0 dynamically deletes the oldest slow SQL queries when the maximum number of slow queries is exceeded. If the specifications of the DRDS instance include 4 cores and 4 GB memory, the system can retain a maximum of 10,000 slow SQL queries. If the specifications of the DRDS instance include 8 cores and 8 GB memory, the system can retain a maximum of 20,000 slow SQL queries. The slow SQL queries include logical slow SQL queries and physical slow SQL queries. A similar rule applies to other instance specifications.

Sample code

```
mysql> show slow where execute_time > 1000 limit 1;
+-----------+---------------------+--------------+------------+-----------+
| HOST      | START_TIME          | EXECUTE_TIME | AFFECT_ROW | SQL       |
+-----------+---------------------+--------------+------------+-----------+
| 127.0.0.1 | 2016-03-16 13:02:57 |         2785 |          7 | show rule |
+-----------+---------------------+--------------+------------+-----------+
1 row in set (0.02 sec)
```

Important columns:

- **HOST** : the IP address of the server from which the SQL statement is sent.

- **START_TIME**: the time when the SQL statement starts to be executed.
- **EXECUTE_TIME**: the time that is spent executing the SQL statement.
- **AFFECT_ROW**: the number of affected rows for a DML statement or the number of returned records for a data query language (DQL) statement.

SHOW [FULL] PHYSICAL_SLOW [WHERE expr] [limit expr]

SQL queries that take more than 1s are slow SQL queries.Physical slow SQL queries are sent from a PolarDB-X 1.0 instance to an ApsaraDB RDS for MySQL instance.

- `SHOW PHYSICAL_SLOW` : queries the top 100 physical slow SQL queries since a PolarDB-X 1.0 instance is started or the last `CLEAR SLOW` statement is executed. Take note that the SHOW PHYSICAL_SLOW statement returns the top 100 physical slow SQL queries.The returned data is stored in PolarDB-X 1.0. If you restart the database instance or execute the CLEAR SLOW statement, the returned data is cleared.
- `SHOW FULL PHYSICAL_SLOW` : queries all physical slow SQL queries that are persistently stored in PolarDB-X 1.0 since the database instance is started. The system can retain a limited number of slow queries. The upper limit varies based on the instance type. PolarDB-X 1.0 dynamically deletes the oldest slow SQL queries when the maximum number of slow queries is exceeded. If the specifications of the DRDS instance include 4 cores and 4 GB memory, the system can retain a maximum of 10,000 slow SQL queries. If the specifications of the DRDS instance include 8 cores and 8 GB memory, the system can retain a maximum of 20,000 slow SQL queries. The slow SQL queries include logical slow SQL queries and physical slow SQL queries. A similar rule applies to other instance specifications.

Sample code

```
mysql> show physical_slow;
+----------------+-----------------------------------+--------------------+--------------+
-----------------+-------------------------+-----------------------+------------+--------
---------+
| GROUP_NAME     | DBKEY_NAME                        | START_TIME          | EXECUTE_TIME |
SQL_EXECUTE_TIME | GETLOCK_CONNECTION_TIME | CREATE_CONNECTION_TIME | AFFECT_ROW | SQL
|
+----------------+-----------------------------------+--------------------+--------------+
-----------------+-------------------------+-----------------------+------------+--------
---------+
| TDDL5_00_GROUP | db218249098_sqa_zmf_tddl5_00_3309 | 2016-03-16 13:05:38 |         1057 |
1011 |                       0 |                       0 |          1 | select sleep(1) |
+----------------+-----------------------------------+--------------------+--------------+
-----------------+-------------------------+-----------------------+------------+--------
---------+
1 row in set (0.01 sec)
```

Important columns:

- **GROUP_NAME**: the name of the group to which the database that executes the SQL statement belongs.
- **START_TIME**: the time when the SQL statement starts to be executed.
- **EXECUTE_TIME**: the time that is spent executing the SQL statement.
- **AFFECT_ROW**: the number of affected rows for a DML statement or the number of returned records for a DQL statement.

CLEAR SLOW

You can execute the CLEAR SLOW statement to clear the top 100 logical slow SQL queries and the top 100 physical slow SQL queries since a PolarDB-X 1.0 instance is started or the last `CLEAR SLOW` statement is executed.

Sample code

```
mysql> clear slow;
Query OK, 0 rows affected (0.00 sec)
```

Note You can execute the `SHOW SLOW` or `SHOW PHYSICAL_SLOW` statement to query the top 100 slow SQL queries. If you do not execute the `CLEAR SLOW` statement for a long period of time, the system may return some invalid slow SQL queries that are optimized. Therefore, we recommend that you execute `CLEAR SLOW` after you optimize slow SQL queries. Then, you can check whether the slow SQL queries are optimized after the system runs for a period of time.

6.4. Statistics queries

This topic describes how to execute the SHOW statements to query real-time statistics.

- SHOW [FULL] STATS
- SHOW DB STATUS
- SHOW FULL DB STATUS [LIKE {tablename}]
- SHOW TABLE STATUS [LIKE 'pattern' | WHERE expr]

SHOW [FULL] STATS

You can execute this SQL statement to query the overall statistics. The statistics are instantaneous values. Take note of this point:The query result of the `SHOW FULL STATS` statement varies based on the PolarDB-X 1.0 instance versions.

Example:

```
mysql> show stats;
+------+---------+----------+------------------+-----------------+----------------------
-+--------------------+--------+------------+--------------+--------------+--------------
-+
| QPS  | RDS_QPS | SLOW_QPS | PHYSICAL_SLOW_QPS | ERROR_PER_SECOND | MERGE_QUERY_PER_SECOND
| ACTIVE_CONNECTIONS | RT(MS) | RDS_RT(MS) | NET_IN(KB/S) | NET_OUT(KB/S) | THREAD_RUNNING
|
+------+---------+----------+------------------+-----------------+----------------------
-+--------------------+--------+------------+--------------+--------------+--------------
-+
| 1.77 |    1.68 |     0.03 |              0.03 |            0.02 |                   0.00
|                  7 | 157.13 |      51.14 |       134.49 |          1.48 |              1
|
+------+---------+----------+------------------+-----------------+----------------------
-+--------------------+--------+------------+--------------+--------------+--------------
-+
1 row in set (0.01 sec)
mysql> show full stats;
+------+---------+----------+------------------+-----------------+----------------------+
------------------------+--------------------+-------------------------------+--------+-----
-------+--------------+---------------+---------------+----------------------+------------
-----+------------------------------+-----------------------+------------------------------+-
------------------------+--------------------------+---------------------+-------+---------
+-------------+------------+
| QPS  | RDS_QPS | SLOW_QPS | PHYSICAL_SLOW_QPS | ERROR_PER_SECOND | VIOLATION_PER_SECOND |
MERGE_QUERY_PER_SECOND | ACTIVE_CONNECTIONS | CONNECTION_CREATE_PER_SECOND | RT(MS) | RDS_R
T(MS) | NET_IN(KB/S) | NET_OUT(KB/S) | THREAD_RUNNING | HINT_USED_PER_SECOND | HINT_USED_CO
UNT | AGGREGATE_QUERY_PER_SECOND | AGGREGATE_QUERY_COUNT | TEMP_TABLE_CREATE_PER_SECOND | T
EMP_TABLE_CREATE_COUNT | MULTI_DB_JOIN_PER_SECOND | MULTI_DB_JOIN_COUNT | CPU   | FREEMEM |
FULLGCCOUNT | FULLGCTIME |
+------+---------+----------+------------------+-----------------+----------------------+
------------------------+--------------------+-------------------------------+--------+-----
-------+--------------+---------------+---------------+----------------------+------------
-----+------------------------------+-----------------------+------------------------------+-
------------------------+--------------------------+---------------------+-------+---------
+-------------+------------+
| 1.63 |    1.68 |     0.03 |              0.03 |            0.02 |                  0.00 |
0.00 |                  6 |                          0.01 | 157.13 |      51.14 |       134.
33 |          1.21 |              1 |                 0.00 |               54 |
0.00 |                  663 |                          0.00 |                      512 |
0.00 |                516 | 0.09% |   6.96% |       76446 |   21326906 |
+------+---------+----------+------------------+-----------------+----------------------+
------------------------+--------------------+-------------------------------+--------+-----
-------+--------------+---------------+---------------+----------------------+------------
-----+------------------------------+-----------------------+------------------------------+-
------------------------+--------------------------+---------------------+-------+---------
+-------------+------------+
1 row in set (0.01 sec)
```

Important columns:

- **QPS**: the number of queries per second (QPS) sent from an applicationto a PolarDB-X 1.0 instance.

The QPS is the logical QPS.

- **RDS_QPS**:the number of QPS sent from a PolarDB-X 1.0 instance to an ApsaraDB RDS for MySQL instance.The QPS is the physical QPS.
- **ERROR_PER_SECOND**: the total number of errors that occur per second. These errors include SQL syntax errors, primary key conflicts, system errors, and connectivity errors.
- **VIOLATION_PER_SECOND**: the number of primary key conflicts or unique key conflicts per second.
- **MERGE_QUERY_PER_SECOND**: the number of queries on tables per second. Sharding is enabled for the DRDS instance.
- **ACTIVE_CONNECTIONS**: the number of active connections.
- **CONNECTION_CREATE_PER_SECCOND**: the number of connections that are created per second.
- **RT(MS)**: the time between a sent SQL query and a response. The SQL query is sent from an application toa PolarDB-X 1.0 instance. The response time (RT) is the logical RT.
- **RDS_RT(MS)**:the time to respond to an SQL query that is sent from a PolarDB-X 1.0 instance to an ApsaraDB RDS for MySQL instance. The RT is the physical RT.
- **NET_IN(KB/S)**:the amount of inbound traffic of a PolarDB-X 1.0 instance per second.
- **NET_OUT(KB/S)**:the amount of outbound traffic of a PolarDB-X 1.0 instance per second.
- **THREAD_RUNNING**: the number of threads that are running in a DRDS instance.
- **HINT_USED_PER_SECOND**: the number of SQL queries that contain hints per second.
- **HINT_USED_COUNT** : the total number of SQL queries that contain hints since a DRDS instance is started.
- **AGGREGATE_QUERY_PER_SECCOND**: the number of aggregate queries per second.
- **AGGREGATE_QUERY_COUNT** : the total number of aggregate queries. This column shows the accumulative historical data.
- **TEMP_TABLE_CREATE_PER_SECCOND**: the number of temporary tables that are created per second.
- **TEMP_TABLE_CREATE_COUNT** : the total number of temporary tables that are created since a DRDS instance is started.
- **MULTI_DB_JOIN_PER_SECCOND**: the number of cross-database JOIN queries per second.
- **MULTI_DB_JOIN_COUNT** : the total number of cross-database JOIN queries since a DRDS instance is started.

SHOW DB STATUS

You can execute this SQL statement to query the storage and performance information about a physical database in real time. The storage information is obtained from an ApsaraDB RDS for MySQL system table. Therefore, the returned storage may be different from the actual storage.

Example:

```
mysql> show db status;
+------+--------------------------+-------------------+------------------+------------+-
-------+----------------+
| ID   | NAME                     | CONNECTION_STRING  | PHYSICAL_DB       | SIZE_IN_MB |
RATIO  | THREAD_RUNNING |
+------+--------------------------+-------------------+------------------+------------+-
-------+----------------+
|    1 | drds_db_1516187088365daui | 100.100.64.1:59077 | TOTAL             |  13.109375 |
100%   | 3              |
|    2 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 |   1.578125 |
12.04% |                |
|    3 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 |     1.4375 |
10.97% |                |
|    4 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0002 |     1.4375 |
10.97% |                |
|    5 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0003 |     1.4375 |
10.97% |                |
|    6 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0004 |   1.734375 |
13.23% |                |
|    7 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0005 |   1.734375 |
13.23% |                |
|    8 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0006 |   2.015625 |
15.38% |                |
|    9 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0007 |   1.734375 |
13.23% |                |
+------+--------------------------+-------------------+------------------+------------+-
-------+----------------+
```

Important columns:

- **NAME**: the internal tag that represents aPolarDB-X 1.0 database.PolarDB-X 1.0 The value is different from the name of the PolarDB-X 1.0 database.
- **CONNECTION_STRING**: the information about a connection from a DRDS instance to a database shard.
- **PHYSICAL_DB**: the name of a database shard. The `TOTAL` row showsthe total storage of all the database shards of a PolarDB-X 1.0 database.
- **SIZE_IN_MB**: the used storage in a database shard. Unit: MB.
- **RATIO**: the ratio of the data volume of a database shard to the total data volume of the PolarDB-X 1.0 database.
- **THREAD_RUNNING**: the number of threads that are running on a physical database instance. The value of the THREAD_RUNNING parameter is the same as that of the Threads_running parameter returned by the `SHOW GLOBAL STATUS` statement in MySQL. For more information, see MySQL official documentation.

SHOW FULL DB STATUS [LIKE {tablename}]

You can execute this SQL statement to query the storage and performance information about a table in a physical database in real time. The storage information is obtained from an ApsaraDB RDS for MySQL system table. Therefore, the returned storage may be different from the actual storage.

Example:

```
mysql> show full db status like hash_tb;
+------+---------------------------+-------------------+------------------+-------------
--+------------+--------+----------------+
| ID   | NAME                      | CONNECTION_STRING  | PHYSICAL_DB       | PHYSICAL_TABL
E | SIZE_IN_MB | RATIO  | THREAD_RUNNING |
+------+---------------------------+-------------------+------------------+-------------
--+------------+--------+----------------+
|    1 | drds_db_1516187088365daui | 100.100.64.1:59077 | TOTAL             |
|     19.875 | 100%   | 3              |
|    2 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 | TOTAL
|    3.03125 | 15.25% |                |
|    3 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 | hash_tb_00
|   1.515625 | 7.63%  |                |
|    4 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0000 | hash_tb_01
|   1.515625 | 7.63%  |                |
|    5 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 | TOTAL
|        2.0 | 10.06% |                |
|    6 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 | hash_tb_02
|   1.515625 | 7.63%  |                |
|    7 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0001 | hash_tb_03
|   0.484375 | 2.44%  |                |
|    8 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0002 | TOTAL
|    3.03125 | 15.25% |                |
|    9 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0002 | hash_tb_04
|   1.515625 | 7.63%  |                |
|   10 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0002 | hash_tb_05
|   1.515625 | 7.63%  |                |
|   11 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0003 | TOTAL
|   1.953125 | 9.83%  |                |
|   12 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0003 | hash_tb_06
|   1.515625 | 7.63%  |                |
|   13 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0003 | hash_tb_07
|     0.4375 | 2.2%   |                |
|   14 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0004 | TOTAL
|    3.03125 | 15.25% |                |
|   15 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0004 | hash_tb_08
|   1.515625 | 7.63%  |                |
|   16 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0004 | hash_tb_09
|   1.515625 | 7.63%  |                |
|   17 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0005 | TOTAL
|   1.921875 | 9.67%  |                |
|   18 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0005 | hash_tb_11
|   1.515625 | 7.63%  |                |
|   19 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0005 | hash_tb_10
|    0.40625 | 2.04%  |                |
|   20 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0006 | TOTAL
|    3.03125 | 15.25% |                |
|   21 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0006 | hash_tb_12
|   1.515625 | 7.63%  |                |
|   22 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0006 | hash_tb_13
|   1.515625 | 7.63%  |                |
|   23 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0007 | TOTAL
|      1.875 | 9.43%  |                |
|   24 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0007 | hash_tb_14
```

```
|   1.515625 | 7.63%  |                |
|   25 | drds_db_1516187088365daui | 100.100.64.1:59077 | drds_db_xzip_0007 | hash_tb_15
|   0.359375 | 1.81%  |                |
+------+---------------------------+--------------------+-------------------+-------------
--+------------+--------+----------------+
```

Important columns:

- **NAME**: the internal tag that represents aPolarDB-X 1.0 database. PolarDB-X 1.0 The value is different from the name of the PolarDB-X 1.0 database.
- **CONNECTION_STRING**: the information about a connection from a DRDS instance to a database shard.
- **PHYSICAL_DB**: the name of a database shard. If you use the LIKE keyword in a statement, the `TOTAL` row shows the storage of the database shard. If you do not use the LIKE keyword in a statement, the TOTAL row shows the total storage of all the database shards.
- **PHYSICAL_TABLE**: the name of a table shard in a database shard. If you use the LIKE keyword in a statement, the `TOTAL` row shows the storage of the table shard. If you do not use the LIKE keyword in a statement, the TOTAL row shows the total storage of all the table shards.
- **SIZE_IN_MB**: the used storage in a table shard. Unit: MB.
- **RATIO**: the ratio of the data volume of a table shard to the total data volume of all the returned table shards.
- **THREAD_RUNNING**: the number of threads that are running on a physical database. The value of the THREAD_RUNNING parameter is the same as that of the Threads_running parameter returned by the `SHOW GLOBAL STATUS` statement in MySQL. For more information, see MySQL official documentation.

SHOW TABLE STATUS [LIKE 'pattern' | WHERE expr]

You can execute this SQL statement to query information about a table. You can use this statement to aggregate the data of all underlying physical table shards.

Example:

```
mysql> show table status like 'multi_db_multi_tbl';
+--------------------+--------+---------+------------+------+----------------+-------------
+-----------------+--------------+-----------+----------------+----------------------+----
---------+------------+-----------------+----------+----------------+----------+
| NAME               | ENGINE | VERSION | ROW_FORMAT | ROWS | AVG_ROW_LENGTH | DATA_LENGTH
| MAX_DATA_LENGTH | INDEX_LENGTH | DATA_FREE | AUTO_INCREMENT | CREATE_TIME          | UPD
ATE_TIME | CHECK_TIME | COLLATION       | CHECKSUM | CREATE_OPTIONS | COMMENT |
+--------------------+--------+---------+------------+------+----------------+-------------
+-----------------+--------------+-----------+----------------+----------------------+----
---------+------------+-----------------+----------+----------------+----------+
| multi_db_multi_tbl | InnoDB |      10 | Compact    |    2 |          16384 |       16384
|               0 |        16384 |         0 |         100000 | 2017-03-27 17:43:57.0 | NUL
L        | NULL       | utf8_general_ci | NULL     |                |         |
+--------------------+--------+---------+------------+------+----------------+-------------
+-----------------+--------------+-----------+----------------+----------------------+----
---------+------------+-----------------+----------+----------------+----------+
1 row in set (0.03 sec)
```

Important columns:

- **NAME**: the name of a table.
- **ENGINE**: the storage engine for a table.
- **VERSION**: the version of a table storage engine.
- **ROW_FORMAT**: the format of the rows in a table. Sample values: Dynamic, Fixed, and Compressed. The Dynamic value specifies that the length of a row is variable, for example, a row of the VARCHAR or BLOB type. The Fixed value specifies that the length of a row is constant, for example, a row of the CHAR or INTEGER type.
- **ROWS**: the number of rows in a table.
- **AVG_ROW_LENGTH**: the average number of bytes in each row.
- **DATA_LENGTH**: the data volume of a full table. Unit: byte.
- **MAX_DATA_LENGTH**: the maximum volume of data that can be stored in a table.
- **INDEX_LENGTH**: the used disk storage by indexes.
- **CREATE_TIME**: the time when a table was created.
- **UPDATE_TIME**: the time when a table was last updated.
- **COLLATION**: the default character set and collation of a table.
- **CREATE_OPTIONS**: the other options specified when you created a table.

You can use the SCAN hint that is provided by PolarDB-X 1.0 in the SHOW TABLE STATUS statement. This way, you can query the data volume of each physical table shard. For more information, see Hints.

```
mysql> /!TDDL:SCAN='multi_db_multi_tbl'*/show table status like 'multi_db_multi_tbl%';
+---------------------+--------+---------+------------+------+----------------+-----------
--+-----------------+--------------+-----------+----------------+----------------------+----
---------+------------+-----------------+----------+----------------+---------+-----------
--+
| Name                | Engine | Version | Row_format | Rows | Avg_row_length | Data_lengt
h | Max_data_length | Index_length | Data_free | Auto_increment | Create_time          | Upd
ate_time | Check_time | Collation       | Checksum | Create_options | Comment | Block_forma
t |
```

```
+----------------------+--------+---------+------------+------+----------------+-----------
--+-----------------+--------------+-----------+----------------+---------------------+----
---------+------------+-----------------+----------+----------------+---------+------------
--+
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    1 |          16384 |       1638
4 |               0 |        16384 |         0 |              2 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L        | NULL       | utf8_general_ci |     NULL |                |         | Original
|
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
```

```
L         | NULL       | utf8_general_ci |     NULL |                |          | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L         | NULL       | utf8_general_ci |     NULL |                |          | Original
|
| multi_db_multi_tbl_1 | InnoDB |      10 | Compact    |    0 |              0 |       1638
4 |               0 |        16384 |         0 |              1 | 2017-03-27 17:43:57 | NUL
L         | NULL       | utf8_general_ci |     NULL |                |          | Original
|
| multi_db_multi_tbl_0 | InnoDB |      10 | Compact    |    1 |          16384 |       1638
4 |               0 |        16384 |         0 |              3 | 2017-03-27 17:43:57 | NUL
L         | NULL       | utf8_general_ci |     NULL |                |          | Original
|
+----------------------+--------+---------+------------+------+----------------+-----------
--+-----------------+--------------+-----------+----------------+---------------------+----
---------+------------+-----------------+----------+----------------+---------+------------
--+
16 rows in set (0.04 sec)
```

6.5. SHOW PROCESSLIST

This topic describes how to use the SHOW PROCESSLIST and SHOW PHYSICAL_PROCESSLIST statements.

SHOW PROCESSLIST

You can execute the following statement to view the connections in PolarDB-X 1.0 and the SQL statements being executed.

- Syntax

```
SHOW PROCESSLIST
```

- Example

```
mysql> SHOW PROCESSLIST\G
     ID: 1971050
   USER: admin
   HOST: 111.111.111.111:4303
     DB: drds_test
COMMAND: Query
   TIME: 0
  STATE:
   INFO: show processlist
1 row in set (0.01 sec)
```

Field	Description
ID	The ID of the connection. The value is a long-type number.
USER	The user name used to establish this connection.

Field	Description
HOST	The IP address and port number of the host that establishes the connection.
DB	The name of the database accessed by this connection.
COMMAND	This field can be set to the either of the following values: ○ Query: The current connection is executing an SQL statement. ○ Sleep: The current connection is idle.
TIME	It is the duration when the connection is in the current state. ○ When the value of COMMAND is Query, this field indicates how long the SQL statement has been executed on the connection. ○ When the value of COMMAND is Sleep, this field indicates how long the connection has been idle.
STATE	This field is meaningless and is constantly empty.
INFO	○ When the value of COMMAND is Query, this field indicates the content of the SQL statement that is being executed on the connection. Note If the FULL parameter is not specified, SHOW PROCESSLIST returns the first 30 characters of each SQL statement that is being executed. If the FULL parameter is specified, SHOW PROCESSLIST returns the first 1,000 characters of each SQL statement that is being executed. ○ If the value of the COMMAND is Sleep, this field is meaningless and is empty.

SHOW PHYSICAL_PROCESSLIST

You can execute the following statement to view information about all physical SQL statements that are being executed.

- Syntax

```
SHOW PHYSICAL_PROCESSLIST
```

Note When an SQL statement in the returned results of the `SHOW PHYSICAL_PROCESSLIST` statement is excessively long, the SQL statement is truncated. In this case, you can execute the `SHOW FULL PHYSICAL_PROCESSLIST` statement to query the complete SQL statement.

- Example

```
mysql> SHOW PHYSICAL_PROCESSLIST\G
*************************** 1. row ***************************
           ID: 0-0-521414
         USER: tddl5
           DB: tddl5_00
      COMMAND: Query
         TIME: 0
        STATE: init
         INFO: show processlist
*************************** 2. row ***************************
           ID: 0-0-521570
         USER: tddl5
           DB: tddl5_00
      COMMAND: Query
         TIME: 0
        STATE: User sleep
         INFO: /*DRDS /88.88.88.88/b67a0e4d8800000/ */ select sleep(1000)
2 rows in set (0.01 sec)
```

Note

- The meaning of each column in the returned results is equivalent to that of the `SHOW PROCESSLIST` statement in MySQL. For more information, see SHOW PROCESSLIST Syntax.
- Different from ApsaraDB RDS for MySQL, the PolarDB-X 1.0 instance returns a string instead of a number in the ID column of a physical connection.

6.6. SHOW GLOBAL INDEX

PolarDB-X 1.0 supports global secondary indexes (GSIs). This topic describes how to use the SHOW GLOBAL INDEX statement to view the GSIs that have been created or are being created.

Syntax

```
SHOW GLOBAL {INDEX | INDEXES} [FROM [schema_name.]tbl_name]
```

`schema_name` and `tbl_name` are optional and are used to filter table names or view table information in other databases.

```
show global index; # Queries the GSIs of all tables in the current database.
show global index from xxx_tb; # Queries the GSI of xxx_tb in the current database.
show global index from xxx_db.xxx_tb; # Queries the GSI of xxx_tb in xxx_db. This is a cros
s-database query.
```

Examples

```
mysql> show global index;
+---------------------+----------------------+------------+------------------------------+
---------------------------+-------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------+------------+------------------+---------------------+----------------
----+------------------+---------------------+--------------------+----------+
| SCHEMA               | TABLE                 | NON_UNIQUE | KEY_NAME                     |
INDEX_NAMES                 | COVERING_NAMES
| INDEX_TYPE | DB_PARTITION_KEY | DB_PARTITION_POLICY | DB_PARTITION_COUNT | TB_PARTITION_K
EY | TB_PARTITION_POLICY | TB_PARTITION_COUNT | STATUS    |
+----------------------+----------------------+------------+------------------------------+
----------------------------+-----------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------+------------+------------------+---------------------+----------------
----+------------------+---------------------+--------------------+----------+
| XXXX_DRDS_LOCAL_APP | full_gsi_ddl_renamed | 1          | g_i_c_ddl_c_blob_long_renamed |
c_blob_long                | id, c_bit_1, c_bit_8, c_bit_16, c_bit_32, c_bit_64, c_tinyint_
1, c_tinyint_1_un, c_tinyint_4, c_tinyint_4_un, c_tinyint_8, c_tinyint_8_un, c_smallint_16,
c_smallint_16_un, c_mediumint_1, c_mediumint_24, c_mediumint_24_un, c_int_1, c_int_32, c_in
t_32_un, c_bigint_1, c_bigint_64, c_bigint_64_un, c_decimal, c_decimal_pr, c_float, c_float
_pr, c_float_un, c_double, c_double_pr, c_double_un, c_date, c_datetime, c_datetime_3, c_da
tetime_6, c_timestamp_1, c_timestamp_3, c_time, c_time_1, c_time_3, c_time_6, c_year, c_yea
r_4, c_char, c_varchar, c_binary, c_varbinary, c_blob_tiny, c_blob_medium, c_text_tiny, c_t
ext, c_text_medium, c_text_long, c_enum, c_set, c_json, c_point, c_linestring, c_polygon, c
_multipoint, c_multilinestring, c_multipolygon, c_geometrycollection, c_geometory
| NULL       | c_blob_long      | HASH                | 4                   | c_blob_long
| HASH                | 3                    | PUBLIC    |
| XXXX_DRDS_LOCAL_APP | full_gsi_ddl_renamed | 1          | g_i_c_ddl_c_mediumint_1       |
c_mediumint_1              | id, c_bit_1, c_bit_8, c_bit_16, c_bit_32, c_bit_64, c_tinyint_
1, c_tinyint_1_un, c_tinyint_4, c_tinyint_4_un, c_tinyint_8, c_tinyint_8_un, c_smallint_16,
c_smallint_16_un, c_mediumint_24, c_mediumint_24_un, c_int_1, c_int_32, c_int_32_un, c_bigi
nt_1, c_bigint_64, c_bigint_64_un, c_decimal, c_decimal_pr, c_float, c_float_pr, c_float_un
, c_double, c_double_pr, c_double_un, c_date, c_datetime, c_datetime_3, c_datetime_6, c_tim
estamp_1, c_timestamp_3, c_time, c_time_1, c_time_3, c_time_6, c_year, c_year_4, c_char, c_
varchar, c_binary, c_varbinary, c_blob_tiny, c_blob_medium, c_blob_long, c_text_tiny, c_tex
t, c_text_medium, c_text_long, c_enum, c_set, c_json, c_point, c_linestring, c_polygon, c_m
ultipoint, c_multilinestring, c_multipolygon, c_geometrycollection, c_geometory, c_smallint
_1, c_timestamp_6 | NULL       | c_mediumint_1    | HASH                | 4
| c_mediumint_1    | HASH                | 3                    | PUBLIC    |
| XXXX_DRDS_LOCAL_APP | full_gsi_ddl_renamed | 1          | g_i_c_ddl_c_smallint_16_un    |
c_smallint_16_un, c_time_1 | id, c_bit_1, c_bit_8, c_bit_16, c_bit_32, c_bit_64, c_tinyint_
1, c_tinyint_1_un, c_tinyint_4, c_tinyint_4_un, c_tinyint_8, c_tinyint_8_un, c_smallint_16,
c_mediumint_1, c_mediumint_24, c_mediumint_24_un, c_int_1, c_int_32, c_int_32_un, c_bigint_
1, c_bigint_64, c_bigint_64_un, c_decimal, c_decimal_pr, c_float, c_float_pr, c_float_un, c
_double, c_double_pr, c_double_un, c_date, c_datetime, c_datetime_3, c_datetime_6, c_timest
amp_1, c_timestamp_3, c_time, c_time_3, c_time_6, c_year, c_year_4, c_char, c_varchar, c_bi
```

```
amp_1, c_timestamp_3, c_time, c_time_3, c_time_6, c_year, c_year_4, c_char, c_varchar, c_bi
nary, c_varbinary, c_blob_tiny, c_blob_medium, c_blob_long, c_text_tiny, c_text, c_text_med
ium, c_text_long, c_enum, c_set, c_json, c_point, c_linestring, c_polygon, c_multipoint, c_
multilinestring, c_multipolygon, c_geometrycollection, c_geometory
| NULL       | c_smallint_16_un | HASH                | 4                  | c_smallint_16_
un | HASH                | 3                  | PUBLIC    |
| XXXX_DRDS_LOCAL_APP | t_order              | 0          | g_i_seller                    |
seller_id                   | id, order_id
| HASH       | seller_id        | HASH                | 4                  | seller_id
| HASH                | 2                  | CREATING |
+--------------------+--------------------+-----------+-----------------------------+
---------------------------+-------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------+------------+------------------+---------------------+---------------
----+------------------+---------------------+--------------------+-----------+
4 rows in set (0.01 sec)
```

List of column names

Column name	Description
SCHEMA	The name of the database.
TABLE	The name of the table.
NON_UNIQUE	Indicates whether the index is a unique GSI. Valid values: • 1: a common GSI • 0: a unique GSI
KEY_NAME	The name of the index.
INDEX_NAMES	The index column.
COVERING_NAMES	The covering column.
INDEX_TYPE	The index type. Valid values: • NULL: not specified • BTREE • HASH
DB_PARTITION_KEY	The database shard key.
DB_PARTITION_POLICY	The database sharding function.
DB_PARTITION_COUNT	The number of database shards.

Column name	Description
TB_PARTITION_KEY	The table shard key.
TB_PARTITION_POLICY	The table sharding function.
TB_PARTITION_COUNT	The number of table shards.
STATUS	The current status of the index. Valid values: • CREATING • DELETE_ONLY • WRITE_ONLY • WRITE_REORG • PUBLIC • ABSENT

6.7. SHOW INDEX

This topic describes how to use the SHOW INDEX statement to view local secondary indexes (LSIs) and global secondary indexes (GSIs) of PolarDB-X 1.0 tables.

Syntax

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [WHERE expr]
```

Examples

```
mysql> show index from t_order;
+--------------+------------+-----------+--------------+----------------+-----------+------
-------+----------+--------+------+------------+----------+---------------+
| TABLE    | NON_UNIQUE | KEY_NAME  | SEQ_IN_INDEX | COLUMN_NAME    | COLLATION | CARDINALIT
Y | SUB_PART | PACKED | NULL | INDEX_TYPE | COMMENT  | INDEX_COMMENT |
+--------------+------------+-----------+--------------+----------------+-----------+------
-------+----------+--------+------+------------+----------+---------------+
| t_order |          0 | PRIMARY   |            1 | id             | A         |
0 |     NULL | NULL   |      | BTREE      |          |               |
| t_order |          1 | l_i_order |            1 | order_id       | A         |
0 |     NULL | NULL   | YES  | BTREE      |          |               |
| t_order |          0 | g_i_buyer |            1 | buyer_id       | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | INDEX    |               |
| t_order |          1 | g_i_buyer |            2 | id             | NULL      |
0 |     NULL | NULL   |      | GLOBAL     | COVERING |               |
| t_order |          1 | g_i_buyer |            3 | order_id       | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
| t_order |          1 | g_i_buyer |            4 | order_snapshot | NULL      |
0 |     NULL | NULL   | YES  | GLOBAL     | COVERING |               |
+--------------+------------+-----------+--------------+----------------+-----------+------
-------+----------+--------+------+------------+----------+---------------+
6 rows in set (0.01 sec)
```

List of column names

Column name	Description
TABLE	The name of the table.
NON_UNIQUE	Indicates whether the index is a unique GSI. Valid values: • 1: a common GSI • 0: a unique GSI
KEY_NAME	The name of the index.
SEQ_IN_INDEX	The sequence number of the index column in the index. The value starts from 1.
COLUMN_NAME	The name of the index column.
COLLATION	The sorting order. Valid values: • A: ascending order • D: descending order • NULL: not sorted
CARDINALITY	The number of estimated unique values.
SUB_PART	The prefix of the index. NULL indicates that the prefix of the index is the entire column.

Column name	Description
PACKED	The information about field compression. NULL indicates no compression.
NULL	Indicates whether the column can be empty.
INDEX_TYPE	The index type. Valid values: • NULL: not specified • BTREE • HASH
COMMENT	The index information. Valid values: • NULL: local index • INDEX: the index column of the GSI • COVERING: the covering column of the GSI
INDEX_COMMENT	Other information of the index.

6.8. SHOW METADATA LOCK

This topic describes how to use the SHOW METADATA LOCK statement in PolarDB-X 1.0 to query a transaction that holds a metadata lock.

Overview

When PolarDB-X 1.0 creates a global secondary index (GSI), it uses a built-in metadata lock to ensure transaction and data consistency. It usually takes a long time to create a GSI for an existing table. If a transaction that holds a metadata lock is running when a GSI is being created, you cannot make changes to the schema before the transaction is completed. In this case, you can use the SHOW METADATA LOCK statement to query the transaction that holds a metadata lock and the corresponding SQL statement that is being executed. This allows you to troubleshoot the long-running transaction that is blocking schema changes.

> **Note** PolarDB-X 1.0 provides the online schema change feature. During the creation of a GSI, the metadata version are switched four times. Two of these switches obtain the write lock of the metadata lock and are immediately unlocked after the metadata is loaded. The write lock is not held for the rest of the time.

Syntax

```
SHOW METADATA {LOCK | LOCKS} [schema_name[.table_name]]
```

`schema_name` and `tbl_name` are optional and are used to filter the displayed database names or table names.

```
show metadata lock; ## Displays all the connections that hold a metadata lock on the node.
show metadata lock xxx_db; # Displays all the connections that hold a metadata lock in xxx_
db on the node.
show metadata lock xxx_db.tb_name; # Displays all the connections that hold a metadata lock
in tb_name of xxx_db on the node.
```

Examples

```
mysql> show metadata lock;
+---------+--------+-----------------+--------------------+-------------+----------------
--+-----------------+----------+-------------------------------------+--------------------
--------------------------+
| CONN_ID | TRX_ID | TRACE_ID        | SCHEMA             | TABLE       | TYPE
| DURATION        | VALIDATE | FRONTEND                            | SQL
|
+---------+--------+-----------------+--------------------+-------------+----------------
--+-----------------+----------+-------------------------------------+--------------------
--------------------------+
| 4       |      0 | f88cf71cbc00001 | XXXX_DRDS_LOCAL_APP | full_gsi_ddl | MDL_SHARED_WRIT
E | MDL_TRANSACTION |        1 | XXXX_DRDS_LOCAL_APP@127.0.0.1:54788 | insert into `full_gs
i_ddl` (id) VALUE (null); |
| 5       |      0 | f88cf71cbc00000 | XXXX_DRDS_LOCAL_APP | full_gsi_ddl | MDL_SHARED_WRIT
E | MDL_TRANSACTION |        1 | XXXX_DRDS_LOCAL_APP@127.0.0.1:54789 | insert into `full_gs
i_ddl` (id) VALUE (null); |
+---------+--------+-----------------+--------------------+-------------+----------------
--+-----------------+----------+-------------------------------------+--------------------
--------------------------+
2 rows in set (0.00 sec)
```

> **Note** This statement is only used to display connections that already hold a metadata lock. It cannot be used to display connections that wait for a metadata lock.

List of column names

Column name	Description
CONN_ID	The ID of the connection that holds the metadata lock.
TRX_ID	The ID of the transaction that holds the metadata lock.
TRACE_ID	The trace ID of the SQL statement that holds the metadata lock.
SCHEMA	The name of the database.
TABLE	The name of the table.
TYPE	The type of the metadata lock that is held.
DURATION	The period for which the metadata lock is held.
VALIDATE	Indicates whether the metadata lock is released.

Column name	Description
FRONTEND	The frontend connection information.
SQL	The SQL statement that holds the metadata lock.

7.DAL

7.1. Manage accounts and permissions

The method to manage accounts and permissions in Distributed Relational Database Service (PolarDB-X 1.0) is the same as that in MySQL. DRDS supports statements such as `GRANT`, `REVOKE`, `SHOW GRANTS`, `CREATE USER`, `DROP USER`, and `SET PASSWORD`.

Accounts

Account description

An account name consists of a username and a hostname. The format is `username@'host'`. If two accounts have the same username but different hostnames, the accounts are each considered to be different accounts. For example, `lily@30.9.73.96` and `lily@30.9.73.100` are two different accounts. The passwords and permissions of the two accounts may be different.

Aftera database is created in the PolarDB-X 1.0 console, the system automatically creates two system accounts for the database: an administrator account and a read-only account. These accounts are built-in accounts. You cannot delete them or modify their permissions.

- The administrator account name is the same as the database name. For example, if the database name is `easydb`, the administrator account name is also `easydb`.
- The read-only account name is the database name suffixed with `_RO`. For example, if the database name is `easydb`, the read-only account name is `easydb_RO`.

For example, two databases are created: `dreamdb` and `andordb`. The `dreamdb` database has an administrator account named `dreamdb` and a read-only account named `dreamdb_RO`. The `andordb` database has an administrator account named `andordb` and a read-only account named `andordb_RO`.

> **Note** Accounts created by executing CREATE USER statements in PolarDB-X 1.0 exist only in PolarDB-X 1.0. These accounts cannot be used in ApsaraDB RDS, so they are not synchronized to ApsaraDB RDS.

Account permissions

- An administrator account has full permissions.
- Only administrator accounts can be used to create other accounts and grant permissions to the created accounts.
- An administrator account is bound to a database and does not have permissions on other databases. The administrator account can only be used to access the database that is bound to the account. You cannot use the administrator account to grant permissions on other databases to other accounts. For example, the easydb administrator account can be used to connect only to the easydb database and can grant permissions only on the easydb database or tables in the easydb database to other accounts.
- A read-only account has only the SELECT permission.

Naming conventions

- An account name is case-sensitive.
- An account name must be 4 to 20 characters in length.

- An account name must start with a letter.
- An account name can contain letters and digits.

Password complexity requirements

- A password must be 6 to 20 characters in length.
- A password can contain letters, digits, and the following special characters: @#$%^&+=

Hostname matching rules

- A hostname must be a value that represents one or more IP addresses. It can contain underscores (`_`) and wildcards (`%`). An underscore (`_`) represents a character and a wildcard (`%`) represents zero or more characters. Hostnames that contain wildcards must be enclosed in single quotation marks (').such as lily@'30.9.%.%' and david@'%'.
- If two accounts in DRDS match the logon user in a host, the account whose hostname contains the longer prefix is the logon account. The name prefix of a host is the IP segment that precedes the wildcards in the IP address of the host. For example, if the `david@'30.9.12_.xxx'` account and the `david@'30.9.1%.234'` account exist in DRDS and the david username is used to log on to the `30.9.127.xxx` host, the logon account is `david@'30.9.12_.xxx'` .
- After Virtual Private Cloud (VPC) is activated, the IP addresses of hosts change.

> **Notice** To prevent invalid account and permission configurations, we recommend that you set the hostname to '%' to match all IP addresses.

Permissions

Support for permissions of different levels

- Database-level permissions are supported.
- Table-level permissions are supported.
- Global permissions are not supported.
- Column-level permissions are not supported.
- Subprogram-level permissions are not supported.

Permission description

Eight basic table permissions are supported: CREATE, DROP, ALTER, INDEX, INSERT, DELETE, UPDATE, and SELECT.

- To execute TRUNCATE statements on a table, you must have the DROP permission on the table.
- To execute REPLACE statements on a table, you must have the INSERT and DELETE permissions on the table.
- To execute CREATE INDEX and DROP INDEX statements, you must have the INDEX permission on the table.
- To execute CREATE SEQUENCE statements, you must have the database-level CREATE permission.
- To execute DROP SEQUENCE statements, you must have the database-level DROP permission.
- To execute ALTER SEQUENCE statements, you must have the database-level ALTER permission.
- To execute INSERT ON DUPLICATE UPDATE statements on a table, you must have the INSERT and UPDATE permissions on the table.

Permission rules

- Permissions are bound to an account (username@'host') instead of a username (username).
- When you grant permissions on a table to an account, the system checks whether the table exists. If the table does not exist, an error is reported.
- The following database account permissions are listed by level in descending order: global permissions, database-level permissions, table-level permissions, and column-level permissions. Global permissions are not supported.
- A granted higher-level permission overwrites lower-level permissions. If you remove the higher-level permission, the lower-level permissions are also removed.
- The USAGE permission is not supported.

Grant permissions on multiple databases to an account

For PolarDB-X 1.0 V5.3.6 or later, the following methods can be used to grant a single account permissions on multiple databases:

- In the Alibaba Cloud PolarDB-X 1.0 console, go to the account management page, create an account, and then grant the required permissions to the account. We recommend that you use this method.
- Execute the CREATE USER statement to create an account, and then execute the GRANT statement to grant the required permissions to the account.

> **Note** If you want to execute SQL statements, pay attention to the following limits:
>
> i. Only administrator accounts can be used to create users and grant them permissions.
>
> ii. An administrator account can only grant permissions on its bound database to other accounts. For example, you create an account named new_user@'%' by using the administrator account of Database A and want to grant the permissions on Database A and Database B to new_user. To meet this demand, you must use the administrator account of Database A to grant the permissions on Database A to new_user and use the administrator account of Database B to grant the permissions on Database B to new_user.

Use an account granted permissions on multiple databases

DRDS V5.3.6 or later allows you to grant a single account the permissions on multiple databases. For example, if an account named new_user@'%' has the SELECT and INSERT permissions on Database A and Database B, pay attention to the following limits when you use this account:

- If you log on to Database A by using the account and you want to query data in Database B, execute the `use B; SELECT * FROM table_in_B;` statement instead of the `SELECT * FROM B.table_in_B;` statement. This is because cross-database queries are not supported.
- If you log on to Database A by using the account and you want to write data to Database B, execute the `use B; INSERT INTO table_in_B VALUES('value');` statement, instead of the `INSERT INTO B.table_in_B VALUES('value');` statement. This is because cross-database data insertion is not supported.
- The same limits also apply to other SQL statements.

Related statements

CREATE USER used to create an account

- Syntax

```
CREATE USER user_specification [, user_specification] ...
user_specification: user [ auth_option ]
auth_option: IDENTIFIED BY 'auth#string'
```

- Examples
  - Create an account named lily@30.9.73.96. The password of the account is 123456. lily is the username. The account can be used to log on to your database only from the host whose IP address is 30.9.73.96.

    ```
    CREATE USER lily@30.9.73.96 IDENTIFIED BY '123456';
    ```

  - Create an account named david@'%'. This account has no password. david is the username. The account can be used to log on to your database from all hosts.

    ```
    CREATE USER david@'%';
    ```

DROP USER used to delete an account

- Syntax

```
DROP USER user [, user] ...
```

- Examples

  Delete the lily@30.9.73.96 account.

```
DROP USER lily@30.9.73.96;
```

SET PASSWORD used to change the password of an account

- Syntax

```
SET PASSWORD FOR user = password_option
password_option: {
    PASSWORD('auth_string')
}
```

- Examples

  Change the password of the lily@30.9.73.96 account to 123456.

```
SET PASSWORD FOR lily@30.9.73.96 = PASSWORD('123456')
```

GRANT used to grant permissions to an account

- Syntax

```
GRANT
    priv_type[, priv_type] ...
    ON priv_level
    TO user_specification [, user_specification] ...
    [WITH GRANT OPTION]
priv_level: {
  | db_name.*
  | db_name.tbl_name
  | tbl_name
}
user_specification:
    user [ auth_option ]
auth_option: {
    IDENTIFIED BY 'auth#string'
}
```

> **Note** If the account specified in the GRANT statement does not exist and no IDENTIFIED BY clause is used, an error message is returned. The error message indicates that the account does not exist. If the account specified in the GRANT statement does not exist but the IDENTIFIED BY clause is used, the specified account is created, and the permissions are granted.

- Examples
  - Create an account named david@'%' for the easydb database. david is the username. The account can be used to log on to the easydb database from all hosts and has full permissions on the easydb database.

    ```
    #Method 1: Execute a statement to create an account, and then execute another statement
    to grant permissions to the account.
    CREATE USER david@'%' IDENTIFIED BY 'your#password';
    GRANT ALL PRIVILEGES ON easydb.* to david@'%';
    #Method 2: Execute only one statement to create an account and grant permissions to the
    account.
    GRANT ALL PRIVILEGES ON easydb.* to david@'%' IDENTIFIED BY 'your#password';
    ```

  - Create an account named hanson@'%' for the easydb database. hanson is the username. The account can be used to log on to the easydb database from all hosts and has full permissions on the easydb.employees table.

    ```
    GRANT ALL PRIVILEGES ON easydb.employees to hanson@'%'
    IDENTIFIED BY 'your#password';
    ```

  - Create an account named hanson@192.168.3.10 for the easydb database. hanson is the username. The account can be used to log on to the easydb database from only 192.168.3.10 and has the INSERT and SELECT permissions on the easydb.emp table.

    ```
    GRANT INSERT,SELECT ON easydb.emp to hanson@'192.168.3.10'
    IDENTIFIED BY 'your#password';
    ```

  - Create a read-only account named actro@'%' for the easydb database. actro is the username. The account can be used to log on to the easydb database from all hosts.

    ```
    GRANT SELECT ON easydb.* to actro@'%' IDENTIFIED BY 'your#password';
    ```

REVOKE used to revoke permissions

- Syntax
  - Delete the permissions at a specific level from an account. The permission level is specified by priv_level.

```
REVOKE
priv_type
[, priv_type] ...
ON priv_level
```

  - Delete the permissions at the database level and the table level from an account.

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM user [, user] ...
```

- Examples
  - Delete the CREATE, DROP, and INDEX permissions on the easydb.emp table from the hanson@'%' account.

```
REVOKE CREATE,DROP,INDEX ON easydb.emp FROM hanson@'%';
```

  - Delete all permissions from the lily@30.9.73.96 account.

```
REVOKE ALL PRIVILEGES,GRANT OPTION FROM lily@30.9.73.96;
```

> **Note** GRANT OPTION must be added to the preceding statement to ensure the compatibility with MySQL.

SHOW GRANTS used to query granted permissions

- Syntax

```
SHOW GRANTS[ FOR user@host];
```

- Examples

```
SHOW GRANTS FOR user1@host;
```

> **Note** In DRDS V5.3.6 and later, the SHOW GRANTS statement can be executed to query the permissions of only the current account. You can log on tothe PolarDB-X 1.0 console to view the information about all accounts and permissions.

7.2. CHECK TABLE

This topic describes how to use the CHECK TABLE statement.

CHECK TABLE checks a table or tables for errors, especially the tables that failed to be created by executing DDL statements.

- For a sharded table, CHECK TABLE checks whether the underlying physical table shards are complete and whether the columns and indexes of each underlying physical table shard are consistent.
- For a single-database non-sharded table, CHECK TABLE checks whether the table exists.

Syntax

```
CHECK TABLE tbl_name
```

Examples

```
mysql>  check table tddl_mgr_log;
+------------------------+-------+----------+----------+
| TABLE                  | OP    | MSG_TYPE | MSG_TEXT |
+------------------------+-------+----------+----------+
| TDDL5_APP.tddl_mgr_log | check | status   | OK       |
+------------------------+-------+----------+----------+
1 row in set (0.56 sec)
mysql> check table tddl_mg;
+-------------------+-------+----------+----------------------------------------+
| TABLE             | OP    | MSG_TYPE | MSG_TEXT                               |
+-------------------+-------+----------+----------------------------------------+
| TDDL5_APP.tddl_mg | check | Error    | Table 'tddl5_00.tddl_mg' doesn't exist |
+-------------------+-------+----------+----------------------------------------+
1 row in set (0.02 sec)
```

7.3. CHECK GLOBAL INDEX

You can execute the CHECK GLOBAL INDEX statement to check whether data is consistent between primary tables and index tables, and modify inconsistent data.

Syntax

```
CHECK GLOBAL INDEX gsi_name [ON tbl_name] [extra_cmd]
```

Parameter	Description
`gsi_name`	The name of the global secondary index (GSI) that needs to be verified.
`tbl_name`	Optional. The primary table where the GSI resides. If you enter the specific name of a primary table, the system checks whether the index relationship between the GSI table and the primary table is valid.
`extra_cmd`	The reserved extra instruction. Valid values: • -: indicates that only the GSI is checked if no keywords are specified. • SHOW: displays the result of the latest verification or correction for the specified GSI table. • CORRECTION_BASED_ON_PRIMARY: corrects data in the GSI table based on the primary table.

> **Note**
>
> - Some system resources are occupied when data in the GSI table is verified or corrected. This occurs especially when data in the primary table or the index table is locked and corrected in batches during the correction operation. We recommend that you perform these operations during off-peak hours. For more information about how to use GSIs, see Use global secondary indexes.
> - It may take a long time to verify the GSIs of large tables. You can use HINT to specify PURE_ASYNC_DDL_MODE to execute data definition language (DDL) statements in pure asynchronous mode. For more information, see Control parameters for DDL execution engine.

Examples

- You can execute the following statement for verification:

```
mysql> CHECK GLOBAL INDEX `g_i_check`;
```

  - If no errors are reported during the verification, the following results are returned:

```
+-------------+------------+--------+-------------+-----------------------------+
| GSI_TABLE   | ERROR_TYPE | STATUS | PRIMARY_KEY | DETAILS                     |
+-------------+------------+--------+-------------+-----------------------------+
| `g_i_check` | SUMMARY    | --     | --          | OK (7025/7025 rows checked) |
+-------------+------------+--------+-------------+-----------------------------+
1 row in set (1.40 sec)
```

  - If errors are reported during the verification, the following results are returned:

```
+-------------+-----------+--------+-------------+-----------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
+
| GSI_TABLE   | ERROR_TYPE | STATUS | PRIMARY_KEY | DETAILS
|
+-------------+-----------+--------+-------------+-----------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
+
| `g_i_check` | ORPHAN     | FOUND  | (100722)    | {"GSI":{"id":100722,"c_timestamp_6"
:"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01 00:00:00.000","c_timestamp_1"
:"2000-01-01 00:00:00.0","c_binary":"OTkAAAAAAAAAA==","c_int_32":271}}
|
| `g_i_check` | CONFLICT   | FOUND  | (108710)    | {"Primary":{"id":108710,"c_timestam
p_6":"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01 00:00:00.000","c_timestam
p_1":"2000-01-01 00:00:00.0","c_year":"2000","c_int_32":255},"GSI":{"c_int_32_un":12345
6,"id":108710,"c_timestamp_6":"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01
00:00:00.000","c_timestamp_1":"2000-01-01 00:00:00.0","c_year":"2000","c_int_32":255}}
|
| `g_i_check` | MISSING    | FOUND  | (100090)    | {"Primary":{"id":100090,"c_timestam
p_6":"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01 00:00:00.000","c_timestam
p_1":"2000-01-01 00:00:00.0","c_blob_tiny":"YeS4reWbvWE=","c_int_32":280}}
|
| `g_i_check` | SUMMARY    | --     | --          | 3 error found (7025/7025 rows check
ed)
|
+-------------+-----------+--------+-------------+-----------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
+
4 rows in set (1.92 sec)
```

> **Note** If data has multiple types of errors, multiple values of ERROR_TYPE are returned for the same row of data.

- You can execute the following statement for correction:

```
mysql> CHECK GLOBAL INDEX g_i_check CORRECTION_BASED_ON_PRIMARY;
```

The following results are returned:

```
+-------------+-----------+--------+-------------+-----------------------------------
---------------------------------+
| GSI_TABLE   | ERROR_TYPE | STATUS | PRIMARY_KEY | DETAILS
|
+-------------+-----------+--------+-------------+-----------------------------------
---------------------------------+
| `g_i_check` | SUMMARY    | --     | --          | Done. Use SQL: { CHECK GLOBAL INDEX `
g_i_check` SHOW; } to get result. |
+-------------+-----------+--------+-------------+-----------------------------------
---------------------------------+
1 row in set (1.40 sec)
```

- You can execute the following statement to view the report of the latest verification or correction:

```
mysql> CHECK GLOBAL INDEX `g_i_check` SHOW;
```

The following results are returned:

```
+-------------+-----------+----------+-------------+-----------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| GSI_TABLE   | ERROR_TYPE | STATUS   | PRIMARY_KEY | DETAILS
|
+-------------+-----------+----------+-------------+-----------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| `g_i_check` | MISSING    | REPAIRED | (100090)    | {"Primary":{"id":100090,"c_timestam
p_6":"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01 00:00:00.000","c_timestamp_
1":"2000-01-01 00:00:00.0","c_blob_tiny":"YeS4reWbvWE=","c_int_32":280}}
|
| `g_i_check` | CONFLICT   | REPAIRED | (108710)    | {"Primary":{"id":108710,"c_timestam
p_6":"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01 00:00:00.000","c_timestamp_
1":"2000-01-01 00:00:00.0","c_year":"2000","c_int_32":255},"GSI":{"c_int_32_un":123456,"i
d":108710,"c_timestamp_6":"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01 00:00:
00.000","c_timestamp_1":"2000-01-01 00:00:00.0","c_year":"2000","c_int_32":255}} |
| `g_i_check` | ORPHAN     | REPAIRED | (100722)    | {"GSI":{"id":100722,"c_timestamp_6"
:"2000-01-01 00:00:00.000000","c_timestamp_3":"2000-01-01 00:00:00.000","c_timestamp_1":"
2000-01-01 00:00:00.0","c_binary":"OTkAAAAAAAAAA==","c_int_32":271}}
|
| `g_i_check` | SUMMARY    | --       | --          | 3 error found (7025/7026 rows check
ed.) Finish time: 2020-01-13 14:41:51.0
|
+-------------+-----------+----------+-------------+-----------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
4 rows in set (0.02 sec)
```

Descriptions of column names

Column name	Description
GSI_TABLE	The name of the GSI.
ERROR_TYPE	The error type. Valid values: • MISSING: missing index • ORPHAN: orphan index • CONFLICT: inconsistent index data • ERROR_SHARD: position error of data shards • SUMMARY: result summary
STATUS	The status. Valid values: • FOUND: An error is found. • REPAIRED: The error has been repaired.
PRIMARY_KEY	The primary key.
DETAILS	The details of the error.

7.4. KILL

This topic describes how to end the execution of an SQL statement in a PolarDB-X 1.0PolarDB-X database by executing the KILL statement.

Prerequisites

You must connect to the PolarDB-X 1.0PolarDB-X database before you can end the execution of an SQL statement in the PolarDB-X 1.0PolarDB-X database by executing the KILL statement. For more information about how to connect to a PolarDB-X 1.0PolarDB-X database, see Connect to database.

Syntax

The syntax of the KILL command supports the following usages.

- You can execute the following statement to end the execution of logical and physical SQL statements on a connection and end the connection.

```
KILL PROCESS_ID
```

> **Note**
> - You can execute the `SHOW [FULL] PROCESSLIST` statement to query `PROCESS_ID`.
> - PolarDB-X 1.0PolarDB-X databases do not support `KILL QUERY` statements.

- You can execute the following statement to end the execution of a specific physical SQL statement.

```
KILL 'PHYSICAL_PROCESS_ID'
```

Example

```
mysql> KILL '0-0-521570';
Query OK, 0 rows affected (0.01 sec)
```

> Note
>
> - You can execute the `SHOW PHYSICAL_PROCESS_ID` statement to query `PHYSICAL_PROCESS_ID`.
> - Values in the `PHYSICAL_PROCESS_ID` column are strings rather than numbers. Therefore, you must enclose the `PHYSICAL_PROCESS_ID` value in this statement with single quotation marks (').

- You can execute the following statement to end the execution of all the physical SQL statements in a PolarDB-X 1.0PolarDB-X database.

```
KILL 'ALL'
```

7.5. USE

The USE statement tells PolarDB-X 1.0 to use the named database as the default database for subsequent operations. This topic describes how to execute the USE statement.

Context

PolarDB-X 1.0 allows you to connect to different databases that are deployed in a PolarDB-X 1.0 instance. This feature is similar to the feature that enables data queries across standalone databases that run on the MySQL engine. When you log on to a PolarDB-X 1.0 instance, use DB_NAME to configure the default database for subsequent operations. You can execute the USE statement to switch between databases. This helps you manage multiple databases at a time.

Note

- Before you switch between databases, ensure that you have the permissions on the databases. You can grant the permissions in the console. For more information, see Account and permission system.
- After you switch to another database, the hints and sequences in original SQL statements take effect on the new database. This rule applies if you do not specify a database in the hints or sequences.

Syntax

```
USE db_name
```

Example

You can execute the following statement to switch to a database named `NEW_DB`:

```
USE NEW_DB
```

8.Sequence

8.1. Overview

This topic introduces the concepts related to sequences. This topic also describes the supported types of sequences.

A sequence provided by Distributed Relational Database Service (PolarDB-X 1.0) generates globally unique numeric values. DRDS sequence values are of the MySQL BIGINT data type that stores signed 64-bit integers. The term DRDS sequence is referred to as sequence in the following description. Sequences are often used to generate globally unique and sequentially incremental numeric values, such as values of primary key columns and values of unique index columns.

Terms

After you understand the following terms, you can select a sequence type that is suitable for your business:

- Consecutive: If the current value in a consecutive sequence is n, the next value must be n + 1. If the next value is not n + 1, the sequence is a nonconsecutive sequence.
- Monotonically increasing: If the current value in a monotonic increasing sequence is n, the next value must be a number greater than n.
- Single point: A single point of failure (SPOF) risk exists.
- Monotonically increasing at the macro level and non-monotonically increasing at the micro level: For example, the values of a sequence can be `1, 3, 2, 4, 5, 7, 6, 8, ...` Such a sequence is monotonically increasing at the macro level and non-monotonically increasing at the micro level.
- Unitization capability: The unitization capability can help you generate numeric sequences that are unique among multiple instances or multiple databases.

Usage

PolarDB-X 1.0 sequences are divided into the following two types:

- Explicit sequence: Use the DDL syntax to create and maintain an explicit sequence. An explicit sequence can be independently used. For example, you can directly modify and query an explicit sequence. You can use `select seq.nextval` to obtain the values in an explicit sequence. `seq` specifies the name of the sequence.
- Implicit sequence: If you specify the AUTO_INCREMENT attribute for a primary key column, an implicit sequence can be used to automatically generate primary key values. PolarDB-X 1.0 automatically maintains the sequence.

Supported types and features of sequences

PolarDB-X 1.0 supports the following four types of sequences.

Type (abbreviation)	Globally unique	Consecutive	Monotonically increasing	Monotonically increasing in the same session	Non-single point	Data type	Readability	Unitization capability
Group sequence (GROUP)	Yes	No	No	Yes	Yes	All integer data types	High	No
Unit group sequence (GROUP)	Yes	No	No	Yes	Yes	All integer data types	High	Yes
Time-based sequence (TIME)	Yes	No	Monotonically increasing at the macro level and non-monotonically increasing at the micro level	Yes	Yes	BIGINT only	Low	No
Simple sequence (SIMPLE)	Yes	Yes	Yes	Yes	No	All integer data types	High	No

Group sequence (GROUP, default sequence type)

A group sequence is a globally unique sequence that provides natural numeric values. Values in a group sequence do not need to be consecutive or monotonically increasing. If you do not specify a sequence type, PolarDB-X 1.0 uses the group sequence type by default.

Implementation mechanism: DRDS uses multiple nodes to generate sequence values. The multi-node model ensures high availability. The system retrieves a segment of values from a database at a time. In scenarios such as network disconnections, not all the values in a segment are used. Therefore, the sequence values are nonconsecutive.

- Advantages: Group sequences are globally unique and prevent SPOFs. Group sequences deliver excellent performance.
- Disadvantages: Group sequences may contain nonconsecutive values and may not start from the

specified start value. The values of group sequences cannot be cyclical.

Unit group sequence (GROUP)

Unit group sequences extend the capabilities of group sequences. A unit group sequence has unitization capabilities and can provide values that are unique among multiple instances or multiple databases. Values in unit group sequences may not be consecutive or monotonically increasing. If only one unit is configured for a unit group sequence, the unit group sequence is equivalent to a common group sequence.

- Advantages: Unit group sequences have all the advantages of group sequences and have unitization capabilities.
- Disadvantages: Unit group sequences may contain nonconsecutive values and may not start from the specified start value. The values of unit group sequences cannot be cyclical.

The way how unit group sequences work is the same as the way how group sequences work. You can use the extension parameters to customize unit indexes and the number of units.

- The number of units determines the global unique sequence space assigned to the unit group sequence.
- A unit index identifies a unit. Each unit occupies a subset of the global unique sequence space.
- If you specify multiple unit indexes, multiple units are specified. The sequence subsets for different units do not overlap. This means that DRDS does not generate the same sequence value for different units.
- You must specify the same number of units and different unit indexes for all the unit group sequences that belong to the same sequence space.

> **Note**
>
> The following versions of DRDS provide unit group sequences:
>
> - V5.2: V5.2.7-1606682 and later. DRDS V5.2.7-1606682 was released on April 27, 2018.
> - V5.3: V5.3.3-1670435 and later. DRDS V5.3.3-1670435 was released on August 15, 2018.

Time-based sequence (TIME)

A time-based sequence value consists of a timestamp, node ID, and serial number. Such a sequence is globally unique and auto-incremental at the macro level. When the values of a time-based sequence are updated, the system does not retrieve values from a database. The sequence values are also not stored as persistent data in the related database. Only the sequence names and types are stored in the database. Time-based sequences deliver excellent performance. For example, the values of a time-based sequence can be `776668092129345536, 776668098018148352, 776668111578333184, 776668114812141568…`

- Advantages: Time-based sequences are globally unique and deliver excellent performance.
- Disadvantages: The values of time-based sequences are nonconsecutive. The START WITH, INCREMENT BY, MAXVALUE, and CYCLE or NOCYCLE parameters are invalid for time-based sequences.

> **Note**
>
> - If a time-based sequence is used for an auto-increment column of a table, the auto-increment column must be of the BIGINT type.
> - The following versions of DRDS provide time-based sequences:
>   - V5.2: V5.2.8-15432885 and later. DRDS V5.2.8-15432885 was released on December 27, 2018.
>   - V5.3: V5.3.6-15439241 and later. DRDS V5.3.6-15439241 was released on December 29, 2018.

Simple sequence (SIMPLE)

Only simple sequences support the INCREMENT BY, MAXVALUE, and CYCLE or NOCYCLE parameters.

- Advantages: Simple sequence values are globally unique, consecutive, and monotonically increasing. Simple sequences provide multiple features. For example, a simple sequence can have a maximum value and the values of a simple sequence can be cyclical.
- Disadvantages: Simple sequences are prone to SPOFs, low performance, and bottlenecks. Use simple sequences with caution.

Each time a simple sequence generates a value, the system stores the value as persistent data in the related database.

Scenarios

The four types of sequences are globally unique and can be used for primary key columns and unique index columns.

- In most scenarios, we recommend that you use group sequences.
- If you need sequence values that are globally unique among multiple instances or multiple databases, you can use unit group sequences.
- In some cases, your business may require the sequence values to be auto-incremental only at the macro level in the overall trend. The values are not necessarily auto-incremental at the micro level. You may also not want the sequence values to be allocated by using the allocation mechanism of a database. In such scenarios, you can use time-based sequences.
- Use only simple sequences for services that have high requirements for consecutive sequence values. Make sure that you understand the low performance of simple sequences.

For example, you can create a sequence that starts from 100000 and has a step size of 1.

- A **simple sequence** generates globally unique, consecutive, and monotonically increasing values, such as `100000, 100001, 100002, 100003, 100004, ..., 200000, 200001, 200002, 200003...` Simple sequence values are persistently stored. Even after services are restarted upon an SPOF, values are still consecutively generated from the breakpoint. However, simple sequences have poor performance because each time a value is generated the system persistently stores the value.
- A **group sequence** or a **unit group sequence** may generate values such as `200001, 200002, 200003, 200004, 100001, 100002, 100003...`

Note

- The start value of a group sequence is not necessarily the same as the value specified by START WITH. A group sequence always starts from a value that is greater than the specified start value. In this example, the specified start value is 100000, but the actual start value of the group sequence is 200001.
- A group sequence provides globally unique values and may contain nonconsecutive values. For example, if a node fails or only some of the values in a segment are used when the connection is closed, the sequence contains nonconsecutive values. In this example, the values are nonconsecutive. The values between 200004 and 100001 are missing.
- You must specify the same number of units and different unit indexes for unit group sequences that belong to the same globally unique sequence. This ensures that the sequence values are unique among multiple instances or multiple databases.

8.2. Limits

This topic describes the limits on using sequences. This topic also describes how to troubleshoot primary key conflicts.

Limits and additional considerations

When you use a sequence, take note of the following points:

- You must specify the START WITH parameter when you convert a sequence from one type to another.
- You cannot convert a unit group sequence to another type or convert a sequence of another type to a unit group sequence. In addition, you cannot change the parameter values of a unit group sequence, except the value of the START WITH parameter.
- Unit group sequences that belong to the same globally unique sequence space must have the same number of units but have different unit indexes.
- Whenthe INSERT statement is executed in a non-sharded Distributed Relational Database Service (PolarDB-X 1.0) database or in a sharded database that has only non-sharded tables but no broadcast tables, PolarDB-X 1.0 automatically optimizes and sends the statement. This way, you do not need to use an optimizer to allocate sequence values. The `INSERT INTO ... VALUES (seq.nextval, ...)` statementis not supported. We recommend that you use the auto-increment column feature provided by ApsaraDB RDS instead.
- If a hint for a specific database shard is used in the INSERT statement such as INSERT INTO ... VALUES ... or INSERT INTO ...SELECT ..., and the destination table uses a sequence, PolarDB-X 1.0 bypasses the optimizer and directly sends the statement for execution. This way, the sequence does not take effect and the auto-increment column in the ApsaraDB RDS table is used to generate IDs in the destination table.
- Make sure that the auto-increment IDs in the same table are generated by using the same method: a sequence or the auto-increment feature provided by ApsaraDB RDS. If the two methods are used in the same table, duplicate IDs may be generated. Duplicate IDs are difficult to identify.
- If you want to use a time-based sequence in an auto-increment column of a table, make sure that the data type of the column is BIGINT.

Troubleshoot primary key conflicts

If a data record is directly written to ApsaraDB RDS and the corresponding primary key value is not a sequence value generated by PolarDB-X 1.0, the primary key value automatically generated by PolarDB-X 1.0 may conflict with the primary key value that corresponds to the data record. To solve this issue, perform the following steps:

1. Execute the `SHOW SEQUENCES` statement to view the existing sequences. Sequences prefixed with AUTO_SEQ_ are implicit sequences. To create an implicit sequence, you must specify the AUTO_INCREMENT parameter in the statement executed to create a table.

   Execute the following statement in your CLI:

   ```
   mysql> SHOW SEQUENCES;
   ```

   The following query result is returned:

   ```
   +--------------------+-------+--------------+------------+-----------+-------+-------+
   | NAME               | VALUE | INCREMENT_BY | START_WITH | MAX_VALUE | CYCLE | TYPE  |
   +--------------------+-------+--------------+------------+-----------+-------+-------+
   | AUTO_SEQ_xkv_t_item | 0     | N/A          | N/A        | N/A       | N/A   | GROUP |
   | AUTO_SEQ_xkv_shard  | 0     | N/A          | N/A        | N/A       | N/A   | GROUP |
   +--------------------+-------+--------------+------------+-----------+-------+-------+
   2 rows in set (0.04 sec)
   ```

2. If the primary key of the xkv_t_item table is ID and duplicate primary key values exist in this table, query the maximum primary key value of this table from PolarDB-X 1.0.

   Execute the following statement in your CLI:

   ```
   mysql> SELECT MAX(id) FROM xkv_t_item;
   ```

   The following query result is returned:

   ```
   +-----------+
   | MAX(id)   |
   +-----------+
   | 8231      |
   +-----------+
   1 row in set (0.01 sec)
   ```

3. Change the maximum sequence value in the table to a value greater than 8231, such as 9000. Then, no error is returned for the subsequently generated auto-increment primary key values when you execute the INSERT statement.

   Execute the following statement in your CLI:

   ```
   mysql> ALTER SEQUENCE AUTO_SEQ_xkv_t_item START WITH 9000;
   ```

8.3. Explicit sequences

Create a sequence

Create a sequence

Group Sequence

- Syntax

```
CREATE [ GROUP ] SEQUENCE <name>
[ START WITH <numeric value> ]
```

- Parameters

Parameter	Description
START WITH	The start value of the group sequence. If you do not specify this parameter, the default value is used. Default value: 100001.

- Examples
  - Method 1

```
mysql> CREATE SEQUENCE seq1;
```

  - Method 2

```
mysql> CREATE GROUP SEQUENCE seq1;
```

Unit group sequence

- Syntax

```
CREATE [ GROUP ] SEQUENCE <name>
[ START WITH <numeric value> ]
[ UNIT COUNT <numeric value> INDEX <numeric value> ]
```

- Parameters

Parameter	Description
START WITH	The start value of the unit group sequence. The default start value depends on the unit index and the number of units. If you do not specify the INDEX parameter or the UNIT COUNT parameter, the default start value is used. Default value: 100001.
UNIT COUNT	The number of units specified for the unit group sequence. Default value: 1.
INDEX	The unit index of the unit group sequence. The value range is from 0 to the value obtained by subtracting 1 from the number of units. Default value: 0.

> **Note**
> - If you do not specify a sequence type, the **group sequence** type is used by default.
> - Values of **group sequences** and **unit group sequences** may be nonconsecutive. The START WITH parameter only provides reference for **group sequences** and **unit group sequences**. A group sequence or a unit group sequence may not start from the value specified by START WITH. The actual start value is always greater than the specified start value.
> - A **group sequence** can be regarded as a special case of **unit group sequences**. A group sequence means that when you create a **unit group sequence**, you set the UNIT COUNT parameter to 1 and the INDEX parameter to 0.

- Examples

  Create a globally unique numeric sequence that has three units. In this example, create three unit group sequences that have the same sequence name, the same number of units, and different unit indexes for three different instances or databases. These three sequences form a globally unique sequence.

  i. Create a unit group sequence for Instance 1 or Database 1.

  ```
  mysql> CREATE GROUP SEQUENCE seq2 UNIT COUNT 3 INDEX 0;
  ```

  ii. Create a unit group sequence for Instance 2 or Database 2.

  ```
  mysql> CREATE GROUP SEQUENCE seq2 UNIT COUNT 3 INDEX 1;
  ```

  iii. Create a unit group sequence for Instance 3 or Database 3.

  ```
  mysql> CREATE GROUP SEQUENCE seq2 UNIT COUNT 3 INDEX 2;
  ```

Time-based Sequence

- Syntax

```
CREATE TIME SEQUENCE <name>
```

> **Notice** The column that is used to store the values of a **time-based sequence** must be of the BIGINT data type.

- Examples

  Execute the following statement in your CLI:

```
7mysql> CREATE TIME SEQUENCE seq3;
```

Simple Sequence

- Syntax

```
CREATE SIMPLE SEQUENCE <name>
[ START WITH <numeric value> ]
[ INCREMENT BY <numeric value> ]
[ MAXVALUE <numeric value> ][ CYCLE | NOCYCLE ]
```

- Parameters

Parameter	Description
START WITH	The start value of the simple sequence. If you do not specify this parameter, the default value is used. Default value: 1.
INCREMENT BY	The increment between two adjacent sequence values. If you do not specify this parameter, the default value is used. Default value: 1.
MAXVALUE	The maximum value allowed by the simple sequence. If you do not specify this parameter, the default value is used. The default maximum value is of the signed BIGINT data type. For example, you can set the maximum value to 9223372036854775807.
CYCLE or NOCYCLE	You can select only CYCLE or NOCYCLE. These options are used to specify whether to repeat the sequence that starts from the value specified by START WITH after the sequence reaches the specified maximum value. If you do not specify an option, the default option is used. Default option: NOCYCLE.

- Examples

  Create a simple sequence. The start value of the simple sequence is 1000, the step size is 2, and the maximum value is 99999999999. After the maximum value is reached, the sequence does not generate values from the start value.

  Execute the following statement in your CLI:

```
mysql> CREATE SIMPLE SEQUENCE seq4 START WITH 1000 INCREMENT BY 2 MAXVALUE 99999999999 NO
CYCLE;
```

Modify a sequence

Distributed Relational Database Service (PolarDB-X 1.0) allows you to modify a sequence in the following ways:

- For a simple sequence, you can change the values of START WITH, INCREMENT BY, MAXVALUE, and CYCLE or NOCYCLE.
- For a group sequence or a unit group sequence, you can change the value of START WITH.
- You can convert a sequence from one type to another, but neither of the source type nor the destination type can be a unit group sequence.

> **Note**
>
> When you convert a sequence from one type to another, take note of the following points:
>
> - The values of group sequences are nonconsecutive. The values of unit group sequences are also nonconsecutive. The value of the START WITH parameter serves only as a reference for these two types of sequences. A group sequence or a unit group sequence may not start from the value specified by START WITH but must start from a value that is greater than the specified value.
> - You cannot convert a sequence from a unit group type to another or from another type to a unit group sequence. In addition, you cannot change the parameter values of a unit group sequence.
> - If you have specified a value for START WITH when you modify a simple sequence, the value takes effect immediately. The next sequence value starts from the specified value. For example, if you change the value of START WITH to 200 when the sequence value increases to 100, the next sequence value starts from 200.
> - Before you change the value of START WITH, analyze the existing sequence values and the rate of generating sequence values to prevent duplicate sequence values from being generated. Exercise caution when you change the value of START WITH.

Group Sequence

- Syntax

```
ALTER SEQUENCE <name> [ CHANGE TO SIMPLE | TIME ]
START WITH <numeric value>
[ INCREMENT BY <numeric value> ]
[ MAXVALUE <numeric value> ]
[ CYCLE | NOCYCLE ]
```

- Parameters

Parameter	Description
START WITH	The start value of the sequence. This parameter has no default value. If this parameter is not specified, it is ignored. This parameter is required when you convert the sequence from one type to another.
INCREMENT BY	The increment between two adjacent simple sequence values. This parameter takes effect only when you convert a group sequence to a simple sequence. If this parameter is not specified, the default value is used. The default value is 1.
MAXVALUE	The maximum value of the simple sequence. This parameter takes effect only when you convert a group sequence to a simple sequence. If this parameter is not specified, the default value is used. The default value is the maximum value of the signed BIGINT type: 9223372036854775807.

Parameter	Description
CYCLE or NOCYCLE	Specifies whether to continue generating sequence values after the sequence value reaches the maximum value of the sequence that starts from the value specified by START WITH. You can specify only one of the two options: CYCLE and NOCYCLE. Specify the CYCLE option to continue generating sequence values. Specify the NOCYCLE option to stop generating sequence values. The CYCLE or NOCYCLE option takes effect only when you convert a group sequence to a simple sequence. If no options are specified, the default option takes effect. The default option is NOCYCLE.

> **Note** When you convert a sequence to a time-based sequence, the preceding parameters are not supported.

Unit group sequences

- Syntax

```
ALTER SEQUENCE <name>
START WITH <numeric value>
```

- Parameters

Parameter	Description
START WITH	The start value of the unit group sequence. This parameter has no default value. If this parameter is not specified, it is ignored.

> **Note** You cannot convert a **unit group sequence** to another type. In addition, you cannot modify the parameters of a unit group sequence.

Time-based Sequence

- Syntax

```
ALTER SEQUENCE <name>[ CHANGE TO GROUP | SIMPLE ]
START WITH <numeric value>
[ INCREMENT BY <numeric value> ]
[ MAXVALUE <numeric value> ]
[ CYCLE | NOCYCLE ]
```

- Parameters

Parameter	Description

Parameter	Description
START WITH	The start value of the sequence. This parameter has no default value. If this parameter is not specified, it is ignored. This parameter is required when you convert the sequence from one type to another.
INCREMENT BY	The increment between two adjacent simple sequence values. This parameter does not take effect when you convert a simple sequence to a group sequence. If this parameter is not specified, the default value is used. The default value is 1.
MAXVALUE	The maximum value of the simple sequence. This parameter does not take effect when you convert a simple sequence to a group sequence. If this parameter is not specified, the default value is used. The default value is the maximum value of the signed BIGINT type: 9223372036854775807.
CYCLE or NOCYCLE	Specifies whether to continue generating sequence values after the sequence value reaches the maximum value of the sequence that starts from the value specified by START WITH. You can specify only one of the two options: CYCLE and NOCYCLE. Specify the CYCLE option to continue generating sequence values. Specify the NOCYCLE option to stop generating sequence values. The CYCLE or NOCYCLE is ineffective when you convert a simple sequence to a group sequence. If no options are specified, the default option takes effect. The default option is NOCYCLE.

Simple Sequence

- Syntax

```
ALTER SEQUENCE <name> [ CHANGE TO GROUP | TIME ]
START WITH <numeric value>
[ INCREMENT BY <numeric value> ]
[ MAXVALUE <numeric value> ]
[ CYCLE | NOCYCLE ]
```

- Parameters

Parameter	Description
START WITH	The start value of the sequence. This parameter has no default value. If this parameter is not specified, it is ignored. This parameter is required when you convert the sequence from one type to another.

Parameter	Description
INCREMENT BY	The increment between two adjacent simple sequence values. This parameter does not take effect when you convert a simple sequence to a group sequence. If this parameter is not specified, the default value is used. The default value is 1.
MAXVALUE	The maximum value of the simple sequence. This parameter does not take effect when you convert a simple sequence to a group sequence. If this parameter is not specified, the default value is used. The default value is the maximum value of the signed BIGINT type: 9223372036854775807.
CYCLE or NOCYCLE	Specifies whether to continue generating sequence values after the sequence value reaches the maximum value of the sequence that starts from the value specified by START WITH. You can specify only one of the two options: CYCLE and NOCYCLE. Specify the CYCLE option to continue generating sequence values. Specify the NOCYCLE option to stop generating sequence values. The CYCLE or NOCYCLE is ineffective when you convert a simple sequence to a group sequence. If no options are specified, the default option takes effect. The default option is NOCYCLE.

Note When you convert a sequence to a time-based sequence, the preceding parameters are not supported.

Convert a sequence from one type to another

When you convert a sequence from one type to another, take note of the following points:

- Use the `CHANGE TO <sequence_type>` clause in the `ALTER SEQUENCE` statement to convert a sequence to another type.
- If you include the `CHANGE TO` clause in `ALTER SEQUENCE`, you must specify the START WITH parameter to prevent duplicate values from being generated. This way, if you forget to specify a start value, duplicate values are not generated. The CHANGE TO clause is optional. If you omit this clause, you do not need to specify the START WITH parameter.
- You cannot convert a unit group sequence to another type or convert a sequence of another type to a unit group sequence.

Examples

- Modify a simple sequence named seq4: Set START WITH to 3000, INCREMENT BY to 5, and MAXVALUE to 1000000, and change NOCYCLE to CYCLE. To modify the simple sequence, execute the following statement:

```
mysql> ALTER SEQUENCE seq4 START WITH 3000 INCREMENT BY 5 MAXVALUE 1000000 CYCLE;
```

- Convert a group sequence to a simple sequence.

```
mysql> ALTER SEQUENCE seq1 CHANGE TO SIMPLE START WITH 1000000;
```

Query the type and value of a sequence

Query a sequence

- Syntax

```
SHOW SEQUENCES
```

- Examples

  Execute the following statement in your CLI:

```
mysql> SHOW SEQUENCES;
```

  The following query result is returned:

```
+------+--------+------------+------------+------------+--------------+------------+-----
--------+-------+--------+
| NAME | VALUE  | UNIT_COUNT | UNIT_INDEX | INNER_STEP | INCREMENT_BY | START_WITH | MAX_
VALUE   | CYCLE | TYPE   |
+------+--------+------------+------------+------------+--------------+------------+-----
--------+-------+--------+
| seq1 | 100000 | 1          | 0          | 100000     | N/A          | N/A        | N/A
| N/A   | GROUP  |
| seq2 | 400000 | 3          | 1          | 100000     | N/A          | N/A        | N/A
| N/A   | GROUP  |
| seq3 | N/A    | N/A        | N/A        | N/A        | N/A          | N/A        | N/A
| N/A   | TIME   |
| seq4 | 1006   | N/A        | N/A        | N/A        | 2            | 1000       | 9999
9999999 | N     | SIMPLE |
+------+--------+------------+------------+------------+--------------+------------+-----
--------+-------+--------+
4 rows in set (0.00 sec)
```

> **Note** In the query result, the values in the TYPE column are the abbreviations of the sequence types.

Query a sequence

- Syntax

```
[<schema_name>.]<sequence name>.NEXTVAL
```

- Examples

- Method 1

```
mysql> SELECT sample_seq.nextval FROM dual;
```

The following query result is returned:

```
+--------------------+
| SAMPLE_SEQ.NEXTVAL |
+--------------------+
|             101001 |
+--------------------+
1 row in set (0.04 sec)
```

- Method 2

Execute the following statement in your CLI:

```
mysql> INSERT INTO some_users (name,address,gmt_create,gmt_modified,intro) VALUES ('sun
',sample_seq.nextval,now(),now(),'aa');
```

> **Note**
>
> - When you use this statement, you include sample_seq.nextval in the SQL statement as a value.
> - If the AUTO_INCREMENT parameter is specified when you create a table, you do not need to specify an auto-increment column when you execute the INSERT statement. PolarDB-X 1.0 automatically manage the value of the AUTO_INCREMENT parameter.

Use the following syntax to query multiple sequence values at a time:

- Syntax

The following code provides the syntax:

```
SELECT [<schema_name>.]<sequence name>.NEXTVAL FROM DUAL WHERE COUNT = <numeric value>
```

- Examples

Execute the following statement in your CLI:

```
mysql> SELECT sample_seq.nextval FROM dual WHERE count = 10;
```

The following query result is returned:

```
+--------------------+
| SAMPLE_SEQ.NEXTVAL |
+--------------------+
|             101002 |
|             101003 |
|             101004 |
|             101005 |
|             101006 |
|             101007 |
|             101008 |
|             101009 |
|             101010 |
|             101011 |
+--------------------+
10 row in set (0.04 sec)
```

Delete a sequence

- Syntax

  Use the following the syntax to delete a sequence:

  ```
  DROP SEQUENCE <name>
  ```

- Examples

  Execute the following statement in your CLI:

  ```
  mysql> DROP SEQUENCE seq3;
  ```

8.4. Implicit sequences

Create a sequence

After you specify the AUTO_INCREMENT attribute for a primary key column in a table shard or a broadcast table, a sequence can be used to automatically generate primary key values.Distributed Relational Database Service (PolarDB-X 1.0) automatically maintains the sequence.

The standard CREATE TABLE syntax is extended, so that you can add the sequence type for an auto-increment column. If you do not specify a type, the default type is used. The default type is GROUP. If a sequence is automatically created by DRDS and associated with a table, the sequence name consists of the `AUTO_SEQ_` prefix and the table name.

Group sequence, time-based sequence, or simple sequence

You can use the following syntax to create a table that uses a **group sequence**, **time-based sequence**, or **simple sequence** for an auto-increment column:

```
CREATE TABLE <name> (
   <column> ... AUTO_INCREMENT [ BY GROUP | SIMPLE | TIME ],
   <column definition>,
   ...
) ... AUTO_INCREMENT=<start value>
```

> **Note** If you set the type to `BY TIME` that represents the **time-based sequence** type, the data type of the specified column must be BIGINT.

Unit group sequence

You can use the following syntax to create a table that use an **unit group sequence**:

```
CREATE TABLE <name> (
   <column> ... AUTO_INCREMENT [ BY GROUP ] [ UNIT COUNT <numeric value> INDEX <numeric val
ue> ],
   <column definition>,
   ...
) ... AUTO_INCREMENT=<start value>
```

Examples

- Example 1: Create a table that uses a group sequence by default to generate values for an auto-increment column.

  ```
  mysql> CREATE TABLE tab1 (
  col1 BIGINT NOT NULL AUTO_INCREMENT,
  col2 VARCHAR(16),
  PRIMARY KEY(col1)
  ) DBPARTITION BY HASH(col1);
  ```

- Example 2: Create three tables. Each table uses an unit group sequence to generate values for an auto-increment column. These three tables are created in three instances or databases. The unit group sequences for the three tables have the same name, the same number of units, and different unit indexes.

  i. Create a table in Instance 1 or Database 1.

  Execute the following statement in your CLI to create a table that uses a unit group sequence and set the unit index of the sequence to 0:

  ```
  mysql> CREATE TABLE tab2 (
  col1 BIGINT NOT NULL AUTO_INCREMENT UNIT COUNT 3 INDEX 0,
  col2 VARCHAR(16),
  PRIMARY KEY(col1)
  ) DBPARTITION BY HASH(col1);
  ```

  ii. Create a table in Instance 2 or Database 2.

  Execute the following statement in your CLI to create a table that uses a unit group sequence and set the unit index of the sequence to 1:

  ```
  mysql> CREATE TABLE tab2 (
  col1 BIGINT NOT NULL AUTO_INCREMENT UNIT COUNT 3 INDEX 1,
  col2 VARCHAR(16),
  PRIMARY KEY(col1)
  ) DBPARTITION BY HASH(col1);
  ```

  iii. Create a table in Instance 3 or Database 3.

  Execute the following statement in your CLI to create a table that uses a unit group sequence and set the unit index of the sequence to 2:

```
mysql> CREATE TABLE tab2 (
col1 BIGINT NOT NULL AUTO_INCREMENT UNIT COUNT 3 INDEX 2,
col2 VARCHAR(16),
PRIMARY KEY(col1)
) DBPARTITION BY HASH(col1);
```

- Example 3: Create a table that uses a time-based sequence to generate values for an auto-increment column.

  Execute the following statement in your CLI to create a table that uses a time-based sequence:

  ```
  mysql> CREATE TABLE tab3 (
  col1 BIGINT NOT NULL AUTO_INCREMENT BY TIME,
  col2 VARCHAR(16),
  PRIMARY KEY(col1)
  ) DBPARTITION BY HASH(col1);
  ```

- Example 4: Create a table that uses a simple sequence to generate values for an auto-increment column.

  Execute the following statement in your CLI to create a table that uses a simple sequence:

  ```
  mysql> CREATE TABLE tab4 (
  col1 BIGINT NOT NULL AUTO_INCREMENT BY SIMPLE,
  col2 VARCHAR(16),
  PRIMARY KEY(col1)
  ) DBPARTITION BY HASH(col1);
  ```

ALTER TABLE

`ALTER TABLE` cannot be used to change the type of a sequence. The following `ALTER TABLE` syntax can be used to change the start value of a sequence:

```
ALTER TABLE <name> ... AUTO_INCREMENT=<start value>
```

> **Note**
>
> - To change the sequence type of a table, execute the `SHOW SEQUENCES` statement to check the sequence name and the sequence type, and then execute the `ALTER SEQUENCE` statement to change the sequence type.
> - After a sequence is used, we recommend that you do not modify the start value specified for the `AUTO_INCREMENT` parameter. If you need to modify the start value, analyze the existing sequence values and the rate of generating sequence values to prevent duplicate sequence values from being generated.

Query the information and sequence types of a table

SHOW CREATE TABLE

The SHOW CREATE TABLE statement returns the type of the sequence that is used to generate values for an auto-increment column in a table shard or a broadcast table.

Use the following syntax to query the information about a table:

```
SHOW CREATE TABLE <name>
```

> **Note**
>
> - `SHOW CREATE TABLE` returns only the type of the sequence and does not return the sequence details. To query the sequence details, execute the `SHOW SEQUENCES` statement.
> - For a table that uses an unit group sequence, `SHOW CREATE TABLE` returns a DDL statement that does not contain the number of units and the unit index of the unit group sequence. Therefore, you cannot execute this DDL statement to create a table and define the table to use an **unit group sequence** that has the same unitization capability of the returned sequence.
> - If you need to create a table and define the table to use an unit group sequence that has the same unitization capability of the sequence of another table, you must execute the `SHOW SEQUENCES` statement to query the number of units and the unit index. Then, modify the DDL statement that is returned by `SHOW CREATE TABLE` based on the `CREATE TABLE` syntax.

Examples

Example 1: When you were creating the tab1 table, you did not specify a sequence type for the `auto-increment` column. A **group sequence** is used by default.

Execute the following statement in your CLI to query the information about the tab1 table:

```
mysql> SHOW CREATE TABLE tab1;
```

The following result is returned:

```
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------+
| tab1  | CREATE TABLE `tab1` (
`col1` bigint(20) NOT NULL AUTO_INCREMENT BY GROUP,
`col2` varchar(16) DEFAULT NULL,
PRIMARY KEY (`col1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`col1`) |
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------+
1 row in set (0.02 sec)
```

Example 2: When you were creating the tab2 table, you specified the number of units and the unit index for an **unit group sequence** that is used for an `auto-increment` column. However, `SHOW CREATE TABLE` does not return the number of units or the unit index. The returned DDL statement cannot be executed to create a table and define the table to use an **unit group sequence** that have the same unitization capability as the sequence used by the tab2 table.

Execute the following statement in your CLI to query the information about the tab2 table:

```
mysql> SHOW CREATE TABLE tab2;
```

The following result is returned:

```
+-------+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
--------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
--------------------------+
| tab2  | CREATE TABLE `tab2` (
`col1` bigint(20) NOT NULL AUTO_INCREMENT BY GROUP,
`col2` varchar(16) DEFAULT NULL,
PRIMARY KEY (`col1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`col1`) |
+-------+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
--------------------------+
1 row in set (0.01 sec)
```

Example 3: When you were creating the tab3 table, you specified a sequence of the `BY TIME` type for an `auto-increment` column. This means that you specified a **time-based sequence**.

Execute the following statement in your CLI to query the information about the tab3 table:

```
mysql> SHOW CREATE TABLE tab3;
```

The following result is returned:

```
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------+
| tab3  | CREATE TABLE `tab3` (
`col1` bigint(20) NOT NULL AUTO_INCREMENT BY TIME,
`col2` varchar(16) DEFAULT NULL,
PRIMARY KEY (`col1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`col1`) |
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------+
1 row in set (0.01 sec)
```

Example 4: When you were creating the tab4 table, you specified a sequence of the `BY SIMPLE` type for an `auto-increment` column. This means that you specified a **simple sequence**.

Execute the following statement in your CLI to query the information about the tab4 table:

```
mysql> SHOW CREATE TABLE tab4;
```

The following result is returned:

```
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------+
| tab3  | CREATE TABLE `tab4` (
`col1` bigint(20) NOT NULL AUTO_INCREMENT BY TIME,
`col2` varchar(16) DEFAULT NULL,
PRIMARY KEY (`col1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`col1`) |
+-------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------+
1 row in set (0.01 sec)
```

SHOW SEQUENCES

You can execute the `SHOW SEQUENCES` statement to query the name and details of the sequences in the tables of a database.

Execute the following statement in your CLI to query the names and details of sequences in the tables of a database:

```
mysql> SHOW SEQUENCES;
```

The following result is returned:

```
+---------------+--------+------------+------------+------------+--------------+-----------
-+---------------------+-------+---------+
| NAME          | VALUE  | UNIT_COUNT | UNIT_INDEX | INNER_STEP | INCREMENT_BY | START_WITH
| MAX_VALUE           | CYCLE | TYPE    |
+---------------+--------+------------+------------+------------+--------------+-----------
-+---------------------+-------+---------+
| seq1          | 100000 | 1          | 0          | 100000     | N/A          | N/A
| N/A                 | N/A   | GROUP   |
| seq2          | 400000 | 3          | 1          | 100000     | N/A          | N/A
| N/A                 | N/A   | GROUP   |
| seq3          | N/A    | N/A        | N/A        | N/A        | N/A          | N/A
| N/A                 | N/A   | TIME    |
| seq4          | 1006   | N/A        | N/A        | N/A        | 2            | 1000
| 99999999999         | N     | SIMPLE  |
| AUTO_SEQ_tab1 | 100000 | 1          | 0          | 100000     | N/A          | N/A
| N/A                 | N/A   | GROUP   |
| AUTO_SEQ_tab2 | 400000 | 3          | 1          | 100000     | N/A          | N/A
| N/A                 | N/A   | GROUP   |
| AUTO_SEQ_tab3 | N/A    | N/A        | N/A        | N/A        | N/A          | N/A
| N/A                 | N/A   | TIME    |
| AUTO_SEQ_tab4 | 2      | N/A        | N/A        | N/A        | 1            | 1
| 9223372036854775807 | N     | SIMPLE  |
+---------------+--------+------------+------------+------------+--------------+-----------
-+---------------------+-------+---------+
8 rows in set (0.01 sec)
```

9.Outline

9.1. Usage notes

This topic describes the outline feature and how to use this feature.

Background

When you use a PolarDB-X 1.0 database, execution plans generated by SQL optimizers may not meet your business requirements. For example, some JOIN or AGGREGATE functions that can be processed by the underlying ApsaraDB RDS for MySQL instances are not pushed down. The outline feature provides a method to specify an execution plan for the SQL statement. You can use hints to create an SQL execution plan, and use the outline feature to ensure that your SQL statement is executed based on the SQL execution plan.

The outline feature allows you to create and manage outlines in the system by executing the CREATE, DROP, RESYNC, DISABLE, ENABLE, and SHOW statements. The following sections describe these statements.

Limits

- An SQL query that includes multiple statements is not supported.
- The question mark (?) cannot be usedas a bind variable in the GROUP BY and ORDER BY clauses.
- In parameter-based match mode, origin_stmt cannot contain constants.
- In parameter-based match mode, the number of bind variables in origin_stmt and the number of bind variables in target_stmt must be the same.
- In exact match mode, target_stmt cannot contain bind variables.
- When you create an outline, origin_stmt cannot be the same as that of an existing outline in the system.
- When you create an outline, the syntax of target_stmt must be correct so that the desired execution plan can be generated.

Create an outline

The CREATE statement is used to create an outline. By default, an outline takes effect after it is created.

```
CREATE outline name ON origin_stmt TO target_stmt
```

Parameter description:

- name indicates the name of the outline that you want to create.
- origin_stmt specifies the SQL statement for which you want to create an outline. If the SQL statement does not contain the question mark (?) variable,an exact match is performed.
- If the SQL statement contains the question mark (?)variable, the SQL statement cannot contain constants and a parameter-based match is performed after the SQL statement is formatted.
- target_stmt is the statement that uses hints to generate a logical plan.

Example 1: Create an outline in exact match mode

```
mysql> create outline t1 on select 1 to select 2;
Query OK, 1 row affected (1.09 sec)
mysql> select 1;
+------+
| ?    |
+------+
|    2 |
+------+
```

When the SQL statement is being executed, the select 1 clause is replaced by the select 2 clause.

Example 2: Create an outline in parameter-based match mode

```
mysql> create outline t2 on select ? to select /*+TDDL:slave()*/ * from ms10 where c1=?;
Query OK, 1 row affected (0.16 sec)
mysql> explain select 1;
+-------------------------------------------------------------------------------+
| LOGICAL PLAN                                                                  |
+-------------------------------------------------------------------------------+
| LogicalView(tables="01.ms10", sql="SELECT * FROM `ms10` WHERE (`c1` = ?)") |
| HitCache:false                                                                |
| UsingOutline: T2                                                              |
+-------------------------------------------------------------------------------+
```

Delete an outline

The DROP statement is used to delete a specified outline.

```
DROP OUTLINE name  #name specifies the name of the outline that you want to delete.
```

Synchronize an outline again

A PolarDB-X 1.0 instance runs on multiple servers. After you create an outline on a server, the outline is synchronized to other servers. An error may occur when the outline is being synchronized. In this case, you must synchronize the outline again.

```
RESYNC OUTLINE name  #name specifies the name of the outline that you want to synchronize a
gain.
```

Disable a specified outline

The DISABLE statement is used to disable a specified outline.

```
DISABLE OUTLINE name #name specifies the name of the outline that you want to disable.
```

Enable a specified outline

The ENABLE statement is used to enable a specified outline.

```
ENABLE OUTLINE name  #name specifies the name of the outline you want to enable.
```

Query the outlines in the system

The SHOW statement is used to query the outlines in the system.

```
SHOW OUTLINES
```

9.2. Error codes

This topic describes the common error codes and the causes of the errors.

- ERR_ORIGIN_STMT_UNEXPECTED_CONST: In parameter-based match mode, origin_stmt contains the constants that are not parameters.
- ERR_PARAM_COUNT_NOT_EQUAL: In parameter-based match mode, the number of bind variables in origin_stmt differs from the number of bind variables in target_stmt.
- ERR_TARGET_STMT_UNEXPECTED_PARAM: In exact match mode, target_stmt contains the bind variables.
- ERR_ORIGIN_STMT_CONFLICTED: origin_stmt used for creating an outline is the same as that of an existing outline in the system.
- ERR_TARGET_STMT_ERROR: target_stmt used for creating an outline fails to generate an execution plan.

10.Prepare SQL

10.1. Introduction to the prepared statement protocol

This topic describes the prepared statement protocol and its support for prepared statements. This topic also describes how to enable the prepared statement protocol in a Java client.

Overview

Distributed Relational Database Service (PolarDB-X 1.0) allows you to enable the client/server binary protocol to execute server-side prepared statements. Prepared statements with placeholders for parameter values have the following benefits:

- Minimized overhead of statement parsing each time a statement is executed. In most cases, database applications process large numbers of near-identical statements in which only a few variable values are different. To execute these near-identical statements in an efficient manner, you need only to change the variable values in a prepared statement.
- Protection against SQL injection attacks.

Description

- Basic information about the prepared statement protocol
  - The protocol supports the following commands:
    - COM_STMT_PREPARE
    - COM_STMT_EXECUTE
    - COM_STMT_CLOSE
    - COM_STMT_RESET
  - The protocol supports Java and other programming languages.
  - For information about the commands supported by prepared statements in MySQL, see Prepared statements.
- All SQL DML statements can be used as prepared statements, such as SELECT, UPDATE, DELETE, and INSERT statements.
- Non-DML SQL statements cannot be used as prepared statements, such as SHOW and SET statements.
- The following statements cannot be used as prepared statements in a MySQL CLI:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
```

Enable the prepared statement protocol in a Java client

- If you want to execute prepared statements in your Java client, forcibly add the useServerPrepStmts=true field to the URL for connecting to MySQL. If you do not add this field, a

regular query is performed.

- Example: jdbc:mysql://xxxxxx:3306/xxxxxx?**useServerPrepStmts=true**

Example in Java:

```
Class.forName("com.mysql.jdbc.Driver");
Connection connection =  DriverManager.getConnection("jdbc:mysql://xxxxxx:3306/xxxxxx?useSe
rverPrepStmts=true", "xxxxx", "xxxxx");
String sql = "insert into batch values(?,?)";
PreparedStatement preparedStatement = connection.prepareStatement(sql);
preparedStatement.setInt(1, 0);
preparedStatement.setString(2, "corona-db");
preparedStatement.executeUpdate();
```

11.Hint

11.1. Overview

This topic describes the usage and the syntax of custom hints.

Information provided in this topic is applicable to PolarDB-X 1.0 V5.3 and later.

Overview

Hints are supplementary to the SQL syntax and play a crucial role in relational databases. Hints allow you to choose execution methods for SQL statements by using the corresponding syntax. This way, you can optimize the execution of SQL statements.

PolarDB-X 1.0 provides special hint syntax. For example, if you already know data is stored in some table shards in specific database shards and you need to route an SQL statement to the database shards for execution, you can use PolarDB-X 1.0 custom hints.

Syntax of PolarDB-X 1.0 custom hints

Syntax

```
/*+TDDL: hint_command [hint_command ...]*/
/!+TDDL: hint_command [hint_command ...]*/
```

Note

- PolarDB-X 1.0 custom hints can be specified in the `/*+TDDL:hint_command*/` format or in the `/!+TDDL:hint_command*/` format.
- A hint is a string that is placed between `/*` and `*/` or between `/!` and `*/`. The string begins with `+TDDL:`. The `hint_command` parameter specifiesone or more PolarDB-X 1.0 custom hint commands that are used to affect specific operations. If you specify multiple hint commands for the `hint_command` parameter, separate them with spaces.
- If you specify custom hints in the `/*+TDDL:hint_command*/` format, PolarDB-X 1.0 add the `-c` parameter to the command used to log on to the MySQL command-line client: mysql. This way, you can execute SQL statements that contain the custom hints on the client. If you do not add the -c parameter, PolarDB-X 1.0 the client deletes comments in SQL statements before it sends the SQL statements to servers for execution. custom hints in this format are defined as MySQL comments. Therefore, PolarDB-X 1.0 the custom hints cannot take effect. For more information, see mysql client options.

Examples

```
# Query the names of physical tables in each database shard.
/*+TDDL:scan()*/SHOW TABLES;
# Route the query to database shard 0000 of a read-only ApsaraDB RDS instance.
/*+TDDL:node(0) slave()*/SELECT * FROM t1;
```

In the examples, `/*+TDDL:scan()*/` and `/*+TDDL:node(0) slave()*/` are custom hints provided by PolarDB-X 1.0. The two hints begin with `+TDDL:`. `scan()`, `node(0)`, and `slave()` are PolarDB-X 1.0 hint commands and are separated with spaces.

Use a hint in an SQL statement

PolarDB-X 1.0 allows you to use hints in DML, DDL, and Data Access Language (DAL) statements. The following list describes the syntax:

- For a DML statement, you can specify a hint at the end of the first keyword of the statement, as shown in the following examples:

```
/*+TDDL: ... */ SELECT ...
/*+TDDL: ... */ INSERT ...
/*+TDDL: ... */ REPLACE ...
/*+TDDL: ... */ UPDATE ...
/*+TDDL: ... */ DELETE ...
/*+TDDL: ... */ CREATE TABLE ...
/*+TDDL: ... */ ALTER TABLE ...
/*+TDDL: ... */ DROP TABLE ...
/*+TDDL: ... */ SHOW ...
...
```

- For DML statements, you can specify a hint at the end of the first keyword of the statements, as shown in the following example:

```
SELECT /*+TDDL: ... */  ...
INSERT /*+TDDL: ... */  ...
REPLACE /*+TDDL: ... */  ...
UPDATE /*+TDDL: ... */  ...
DELETE /*+TDDL: ... */  ...
...
```

> **Note** Different hints may be applicable to different statements. For more information about the applicable statements, see the topics that describe hint commands.

Use multiple hints in an SQL statement

PolarDB-X 1.0 allows you to use a hint that contains multiple hint commands in an SQL statement.

```
SELECT /*+TDDL:node(0) slave()*/ ...;
```

PolarDB-X 1.0 has the following limits on the use of hints that contain multiple hint commands:

```
# A single SQL statement cannot contain multiple hints.
SELECT /*+TDDL:node(0)*/ /*+TDDL:slave()*/ ...;
# A hint cannot contain duplicate hint commands.
SELECT /*+TDDL:node(0) node(1)*/ ...;
```

Categories of PolarDB-X 1.0 custom hints

- Read/write splitting
- Specify a custom time-out period for an SQL statement
- Specify database shards where an SQL statement is to be executed
- Scan all or some of the table shards in all or some of the database shards

11.2. Read/write splitting

This topic describes hints for read/write splitting.

Information provided in this topic is applicable to PolarDB-X 1.0 V5.3 and later.

PolarDB-X 1.0 provides read/write splitting that is transparent to the application layer. A latency of several milliseconds exists when data is synchronized between primary and read-only ApsaraDB RDS instances. If you need to read data changes immediately after the data in the primary ApsaraDB RDS instance is changed, you must make sure that the SQL request for reading data is routed to the primary ApsaraDB RDS instance. To meet this demand, PolarDB-X 1.0 provides custom hints for read/write splitting. These custom hints allow you to route SQL statements to a specified primary or read-only ApsaraDB RDS instance.

Syntax

```
/*+TDDL:
    master()
    | slave()
*/
```

You can use the custom hints to specify whether to execute an SQL statement on a primary or read-only ApsaraDB RDS instance. If you use `/*+TDDL:slave()*/` in an SQL statement and a primary ApsaraDB RDS instance is associated with multiple read-only ApsaraDB RDS instances, PolarDB-X 1.0 randomly selects a read-only ApsaraDB RDS instance based on the assigned weight to execute the SQL statement.

Note

- PolarDB-X 1.0 custom hints can be specified in the `/*+TDDL:hint_command*/` format or in the `/!+TDDL:hint_command*/` format.
- If you specify custom hints in the `/*+TDDL:hint_command*/` format, PolarDB-X 1.0 add the `-c` parameter to the command used to log on to the MySQL command-line client: mysql. This way, you can execute SQL statements that contain the PolarDB-X 1.0 custom hints on the client. If you do not add the -c parameter, PolarDB-X 1.0 the client deletes comments in SQL statements before it sends the SQL statements to servers for execution. PolarDB-X 1.0 custom hints in this format are defined as MySQL comments. Therefore, PolarDB-X 1.0 the custom hints cannot take effect. For more information, see mysql client options.

Examples

- Execute an SQL statement on your primary ApsaraDB RDS instance:

  ```
  SELECT /*+TDDL:master()*/ * FROM table_name;
  ```

  After the custom hint `/*+TDDL:master()*/` is added at the end of the first keyword in the SQL statement, this SQL statement is routed to the primary ApsaraDB RDS instance.

- Execute an SQL statement on a specified read-only ApsaraDB RDS instance:

  ```
  SELECT /*+TDDL:slave()*/ * FROM table_name;
  ```

After the custom hint `/*+TDDL:slave()*/` is added at the end of the first keyword in the SQL statement, this SQL statement is randomly routed to a read-only ApsaraDB RDS instance based on the assigned weight.

> **Note**
>
> - The custom hints for read/write splitting are only applicable to read-only SQL statements that are not included in transactions. SQL statements that are executed to write data or are included in transactions are still routed to the primary ApsaraDB RDS instance.
> - If you use the `/*+TDDL:slave()*/` hint in an SQL statement, PolarDB-X 1.0 randomly routes the SQL statement to a read-only ApsaraDB RDS instance based on the assigned weight. If no read-only ApsaraDB RDS instances are available, no error is reported. Instead, the primary ApsaraDB RDS instance is selected to execute the SQL statement.

11.3. Specify a custom time-out period for an SQL statement

This topic describes the hint syntax for specifying a time-out period for SQL statements.

Information provided in this topic is applicable to PolarDB-X 1.0 V5.3 and later.

In PolarDB-X 1.0, the default time-out period for the SQL statements in PolarDB-X 1.0 instances and ApsaraDB RDS instances is 900 seconds. You can change the time-out period based on your requirements. Some SQL statements may take longer than 900 seconds to complete. For these slow SQL statements, PolarDB-X 1.0 provides a custom hint that you can use to change the time-out period for these statements. You can use this custom hint to change the time-out period for SQL statements.

Syntax

```
/*+TDDL:SOCKET_TIMEOUT(time)*/
```

The value specified by `SOCKET_TIMEOUT` is measured in milliseconds. You can use this custom hint to change the time-out period for SQL statements based on your business requirements.

Note

- PolarDB-X 1.0 custom hints can be specified in the `/*+TDDL:hint_command*/` format or in the `/!+TDDL:hint_command*/` format.
- If you specify custom hints in the `/*+TDDL:hint_command*/` format, add the `-c` parameter to the command used to log on to the MySQL command-line client: mysql. This way, you can execute SQL statements that contain the PolarDB-X 1.0 custom hints on the client. If you do not add the -c parameter, the client deletes comments in SQL statements before it sends the SQL statements to servers for execution. PolarDB-X 1.0 custom hints in this format are defined as MySQL comments. Therefore, the PolarDB-X 1.0 custom hints cannot take effect. For more information, see mysql client options.

Examples

Set the time-out period of SQL statements to 40 seconds:

```
/*+TDDL:SOCKET_TIMEOUT(40000)*/SELECT * FROM t_item;
```

Note A longer timeout period causes database resources to be occupied for a longer period of time. If excessive SQL statements are executed over a long time within the same period, a large number of database resources may be consumed, and DRDS database services cannot be provided as expected. To resolve the issue, you must optimize SQL statements that take a long time to execute if possible.

11.4. Specify database shards where an SQL statement is to be executed

This topic describes the hint syntax that is used to specify one or more database shards on which you want to execute an SQL statement. This topic also provides some sample code.

Information provided in this topic is applicable to PolarDB-X 1.0 V5.3 and later.

When you are using PolarDB-X 1.0, if you encounter anSQL statement that is not supported by PolarDB-X 1.0, you can use `NODE HINT` provided by PolarDB-X 1.0. NODE HINT can route an SQL statement to one or more database shards on which you want to execute the SQL statement. If you need to query the data in a specified database shard or in a specified table shard of a database shard, you can use `NODE HINT` to route the SQL statement to the database shard.

Syntax

`NODE HINT` allows you to specify the names of the database shards on which you want to execute an SQL statement. A shard name isthe unique identifier of a database shard in a PolarDB-X 1.0 database. To query the names of the database shards in a database, you can execute the `SHOW NODE` statement.

You can use two methods to specify the names of the database shards on which an SQL statement is executed. One of the methods is to specify only one database shard on which you want to execute the SQL statement. The other method is to specify multiple database shards on which you want to execute the SQL statement.

Notice If the hint statement that is used to specify database shards is contained in an INSERT statement and this INSERT statement contains a sequence definition for the table on which the SQL statement is executed, the sequence does not take effect. For more information, see Limits.

- Specify a database shard on which you want to execute an SQL statement.

  ```
  /*+TDDL:node('node_name')*/
  ```

  `node_name` specifies the name of a database shard. You can customize a PolarDB-X 1.0 hint to route an SQL statement to a database shard that is specified by `node_name` for execution.

- Specify multiple database shards on which you want to execute an SQL statement.

  ```
  /*+TDDL:node('node_name'[,'node_name1','node_name2'])*/
  ```

You can specify multiple shard names and separate the shard names with commas (,). The SQL statement that contains the specified hint is routed to the specified database shards.

> **Note**
> - When you execute an SQL statement that contains a DRDS hint, PolarDB-X 1.0 routes the SQL statement to the database shards for execution. The table names that are specified in the SQL statement must exist in the specified database shards.
> - `NODE HINT` can be used in DML statements, DDL statements, and Data Access Language (DAL) statements.

Note

- In DRDS V5.4.1 and later, PolarDB-X 1.0 adds a four-character random string to each of the names of the physical tables that correspond to table shards. You must execute the SHOW TOPOLOGY statement to obtain the topological relationships of logical tables and the names of physical tables.
- In DRDS V5.4.4 and later, PolarDB-X 1.0 provides a switch to control whether the name of each physical table for table shards contains a random string. By default, this switch is turned on. To turn off the switch, you can log on to the DRDS console and then click the ID of the instance that you want to manage. In the left-side navigation pane of the instance details page, click Parameter Settings. On the page that appears, click the Database tab, and set the value of ENABLE_RANDOM_PHY_TABLE_NAME to false. You can also use the following hint to turn off the switch for the tables that are specified in an SQL statement: `/*+TDDL:cmd_extra(ENABLE_RANDOM_PHY_TABLE_NAME=FALSE)*/`.
- You can specify PolarDB-X 1.0 hints in the following formats: `/*+TDDL:hint_command*/` and `/!+TDDL:hint_command*/`.
- If you specify hints in the `/*+TDDL:hint_command*/` format, add the `-c` parameter to the command used to log on to the MySQL command-line client: mysql. This way, you can execute SQL statements that contain the PolarDB-X 1.0 hints on the client. If you do not add the -c parameter, the client deletes comments in the SQL statements before it sends the SQL statements to servers for execution. PolarDB-X 1.0 hints in this format are defined as MySQL comments. Therefore, the PolarDB-X 1.0 hints are deleted and cannot take effect. For more information, see mysql client options.

Examples

The following example shows the result that is returned by `SHOW NODE` for a PolarDB-X 1.0 database named `drds_test`:

```
mysql> SHOW NODE\G
*************************** 1. row ******************
                 ID: 0
               NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS
  MASTER_READ_COUNT: 212
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 2. row ******************
                 ID: 1
               NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0001_RDS
  MASTER_READ_COUNT: 29
   SLAVE_READ_COUNT: 0
```

```
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 3. row ******************
               ID: 2
             NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0002_RDS
  MASTER_READ_COUNT: 29
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 4. row ******************
               ID: 3
             NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS
  MASTER_READ_COUNT: 29
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 5. row ******************
               ID: 4
             NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0004_RDS
  MASTER_READ_COUNT: 29
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 6. row ******************
               ID: 5
             NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0005_RDS
  MASTER_READ_COUNT: 29
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 7. row ******************
               ID: 6
             NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS
  MASTER_READ_COUNT: 29
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
*************************** 8. row ******************
               ID: 7
             NAME: DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0007_RDS
  MASTER_READ_COUNT: 29
   SLAVE_READ_COUNT: 0
MASTER_READ_PERCENT: 100%
 SLAVE_READ_PERCENT: 0%
8 rows in set (0.02 sec)
```

The result shows that each database shard has the `NAME` attribute. This attribute indicates the name of the database shard. Each shard name in the returned result corresponds to one unique database shard. For example, the shard name `DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0003_RDS` corresponds to the database shard `drds_test_vtla_0003`. After you obtain the shard names, you can use a PolarDB-X 1.0 hint to specify the database shards on which you want to execute an SQL statement.

- Execute an SQL statement on database shard 0.

```
SELECT /*TDDL:node('DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS')*/ * FROM table_name;
```

- Execute an SQL statement on multiple database shards.

```
SELECT /*TDDL:node('DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS','DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS')*/ * FROM table_name;
```

  The SQL statement is executed on the `DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS` shard and the `DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0006_RDS` shard.

- View the physical execution plan of an SQL statement on database shard 0.

```
/*TDDL:node('DRDS_TEST_1473471355140LRPRDRDS_TEST_VTLA_0000_RDS')*/ EXPLAIN SELECT * FROM table_name; ```
```

11.5. Scan all or some of the table shards in all or some of the database shards

This topic describes the hint syntax and sample code that can be used to scan all or some of the table shards in all or some of the database shards.

Information provided in this topic is applicable to PolarDB-X 1.0 V5.3 and later.

DRDS provides the capability to route an SQL statement to one or more database shards for execution. PolarDB-X 1.0 also provides `SCAN HINT` to scan all or some of the table shards in all or some of the database shards. You can use `SCAN HINT` to route an SQL statement to all database shards at a time. For example, you can query all the table shards in a specified database shard or query the amount of the data in each physical table that corresponds to a specified logical table.

You can use `SCAN HINT` to execute SQL statements in the following manners:

1. Execute an SQL statement on all the table shards in all database shards.
2. Execute an SQL statement on all the table shards in the specified database shards.
3. Execute an SQL statement on the specified table shards in the specified database shards. The names of the physical tables are calculated based on the given conditions.
4. Execute an SQL statement on the specified table shards in the specified database shards. The table shards are explicitly specified by using the names of the physical tables.

`SCAN HINT` can be used in DML statements, DDL statements, and some Data Access Language (DAL) statements.

Syntax

```
# SCAN HINT
# Route an SQL statement to all the table shards in all database shards.
SCAN()
# Route an SQL statement to all the table shards in the specified database shards.
SCAN(NODE="node_list")               # Specify the database shards.
# Route an SQL statement to the specified table shards in the specified database shards. Th
e names of the physical tables are calculated based on the given conditions.
SCAN(
  [TABLE=]"table_name_list"          # Specify the names of the logical tables.
  , CONDITION="condition_string"     # Calculate the names of the physical tables based on
the values of the TABLE and CONDITION parameters.
  [, NODE="node_list"] )             # Filter the results obtained based on the value of th
e CONDITION parameter to retain only the names of the tables that are in the specified phys
ical databases.
# Route an SQL statement to the specified table shards in the specified database shards. Th
e table shards are explicitly specified by using the names of the physical tables.
SCAN(
  [TABLE=]"table_name_list"          # Specify the names of the logical tables.
  , REAL_TABLE=("table_name_list")   # Specify the names of the physical tables. These phys
ical table names are used to query data from all the physical databases.
  [, NODE="node_list"] )             # Filter the results obtained based on the value of th
e CONDITION parameter to retain only the names of the tables that are in the specified phys
ical databases.
# Specify the names of physical table shards or logical tables.
table_name_list:
    table_name [, table_name]...
# Specify physical databases by using GROUP_KEY and GROUP_INDEX. To obtain the values of GR
OUP_KEY and GROUP_INDEX, you can execute the SHOW NODE statement.
node_list:
    {group_key | group_index} [, {group_key | group_index}]...
# Specify an SQL WHERE clause. You must specify conditions for each table, such as t1.id =
2 and t2.id = 2.
condition_string:
    where_condition
```

Note

- You can specify PolarDB-X 1.0 hints in the following formats: `/*+TDDL:hint_command*/` and `/!+TDDL:hint_command*/`.
- If you specify hints in the `/*+TDDL:hint_command*/` format, add the `-c` parameter to the command used to log on to the MySQL command-line client: mysql. This way, you can execute SQL statements that contain the PolarDB-X 1.0 hints on the client. If you do not add the -c parameter, the client deletes comments in the SQL statements before it sends the SQL statements to servers for execution. PolarDB-X 1.0 hints in this format are defined as MySQL comments. Therefore, the PolarDB-X 1.0 hints are deleted and cannot take effect. For more information, see mysql client options.

Examples

- Execute an SQL statement on all the table shards in all database shards.

```
SELECT /*+TDDL:scan()*/ COUNT(1) FROM t1
```

When the SQL statement is executed, DRDS routes the SQL statement to all the physical tables of logical table `t1`. Then, DRDS merges the result sets and returns the final result.

- Execute an SQL statement on all the table shards in the specified database shards.

```
SELECT /*+TDDL:scan(node='0,1,2')*/ COUNT(1) FROM t1
```

When the SQL statement is executed, DRDS calculates the names of the physical tables of logical table `t1` in database shards 0000, 0001, and 0002. Then, DRDS routes the SQL statement to the table shards. After the SQL statement is executed, DRDS merges the result sets and returns the final result.

- Execute an SQL statement on the specified table shards based on the given conditions.

```
SELECT /*+TDDL:scan('t1', condition='t1.id = 2')*/ COUNT(1) FROM t1
```

When the SQL statement is executed, DRDS calculates the names of all the physical tables correspond to logical table `t1` and meet the `conditions`. Then, DRDS routes the SQL statement to the specified table shards. After the SQL statement is executed, DRDS merges the result sets and returns the final result.

- Execute an SQL statement that contains a JOIN clause on the specified table shards based on the given conditions.

```
SELECT /*+TDDL:scan('t1, t2', condition='t1.id = 2 and t2.id = 2')*/ * FROM t1 a JOIN t2
b ON a.id = b.id WHERE b.name = "test"
```

When the SQL statement is executed, DRDS calculates the names of the physical tables correspond to logical tables `t1` and `t2` and meet the `conditions`. Then, DRDS routes the SQL statement to the specified table shards. After the SQL statement is executed, DRDS merges the result sets and returns the final result. **Note**: When you use this hint, you must make sure that the two tables belong to the same database shard. You must also make sure that the number of shards in one of the table is the same as the number of shards in the other table. Otherwise, the table shard names obtained by PolarDB-X 1.0 represent table shards that are not in the same database shard. If this issue occurs, DRDS reports an error.

- Execute an SQL statement on the specified table shards in the specified database shards. The table shards are explicitly specified by using the names of the physical tables.

```
SELECT /*+TDDL:scan('t1', real_table=("t1_00", "t1_01"))*/ COUNT(1) FROM t1
```

When the SQL statement is executed, DRDS routes the SQL statement to table shards `t1_00 and t1_01` in all database shards. After the SQL statement is executed, DRDS merges the result sets and returns the final result.

- Execute an SQL statement that contains a JOIN clause on the specified table shards in the specified database shards. The table shards are explicitly specified by using the names of the physical tables.

```
SELECT /*+TDDL:scan('t1, t2', real_table=("t1_00,t2_00", "t1_01,t2_01"))*/ * FROM t1 a JO
IN t2 b ON a.id = b.id WHERE b.name = "test";
```

When the SQL statement is executed, DRDS routes the SQL statement to table shards `t1_00`, `t2_00`, `t1_01`, and `t2_01` in all database shards. After the SQL statement is executed, DRDS merges the result sets and returns the final result.

11.6. Automatic protection against high-risk SQL statements

To prevent data loss due to misoperations, PolarDB-X 1.0 prohibits high-risk SQL statements by default, such as a DELETE statement without a WHERE or LIMIT clause and an UPDATE statement without a WHERE or LIMIT clause. If you need to perform full-table deletion or update, you can skip the preceding protection by adding a hint to the corresponding statement.

Statements

You can add the following hint to an UPDATE or DELETE statement to perform full-table deletion or update:

```
/! TDDL:FORBID_EXECUTE_DML_ALL=false*/
```

Examples

- If a DELETE statement does not contain any WHERE or LIMIT clauses, the execution of this statement is intercepted and the following error message appears:

  ```
  DELETE FROM tt;
  ERR-CODE: [TDDL-4620][ERR_FORBID_EXECUTE_DML_ALL] Forbid execute DELETE ALL or UPDATE ALL
  sql. More: [http://
  example.aliyundoc.com/faq/faqByFaqCode.html?faqCode=TDDL-4620]
  ```

  The operation is successful after the following hint is added to the statement:

  ```
  /!TDDL:FORBID_EXECUTE_DML_ALL=false*/DELETE FROM tt;
  Query OK, 10 row affected (0.21 sec)
  ```

- If an UPDATE statement does not contain any WHERE or LIMIT clauses, the execution of this statement is intercepted and the following error message appears:

  ```
  UPDATE tt SET id = 1;
  ERR-CODE: [TDDL-4620][ERR_FORBID_EXECUTE_DML_ALL] Forbid execute DELETE ALL or UPDATE ALL
  sql. More: [http://example.aliyundoc.com/faq/faqByFaqCode.html?faqCode=TDDL-4620]
  ```

  The operation is successful after the following hint is added to the statement:

  ```
  /! TDDL:FORBID_EXECUTE_DML_ALL=false*/UPDATE tt SET id = 1;
  Query OK, 10 row affected (0.21 sec)
  ```

11.7. INDEX HINT

PolarDB-X 1.0 supports global secondary indexes (GSIs). This topic describes how to use the INDEX HINT command to obtain query results from a specified GSI.

Limits

- The version of the ApsaraDB RDS for MySQL instance must be 5.7 or later, and the version of the PolarDB-X 1.0 instance must be 5.4.1 or later.
- The INDEX HINT command takes effect only for SELECT statements.

Precautions

Custom PolarDB-X 1.0 hints can be in the formats of `/*+TDDL:hint_command*/` and `/!+TDDL:hint_command*/`. If you use the `/*+TDDL:hint_command*/` format, add the -c parameter to the logon command when you use the MySQL command-line client to execute an SQL statement that contains a custom PolarDB-X 1.0 hint. Otherwise, the client deletes the MySQL comment, which represents the custom PolarDB-X 1.0 hint, from the SQL statement and then sends the statement to the server for execution. As a result, the custom PolarDB-X 1.0 hint becomes invalid. For more information, see MySQL client options.

Syntax

PolarDB-XPolarDB-X 1.0 supports the following two types of hint syntax:

- `FORCE INDEX()`: Its syntax is the same as that of MySQL FORCE INDEX. If the specified index is not a GSI, the FORCE INDEX hint is sent to the ApsaraDB RDS for MySQL instance for execution.

```
# FORCE INDEX()
tbl_name [[AS] alias] [index_hint]
index_hint:
    FORCE INDEX({index_name})
```

- `INDEX()`: It specifies a GSI based on the combination of the table name and index name or the combination of the table alias in the current query block and the index name.

```
# INDEX()
/*+TDDL:
    INDEX({table_name | table_alias}, {index_name})
*/
```

> **Note** The preceding statement does not take effect in the following scenarios:
> - The query does not contain the specified table name or alias.
> - The specified GSI is not in the specified table.

Examples

```
CREATE TABLE t_order (
 `id` bigint(11) NOT NULL AUTO_INCREMENT,
 `order_id` varchar(20) DEFAULT NULL,
 `buyer_id` varchar(20) DEFAULT NULL,
 `seller_id` varchar(20) DEFAULT NULL,
 `order_snapshot` longtext DEFAULT NULL,
 `order_detail` longtext DEFAULT NULL,
 PRIMARY KEY (`id`),
 GLOBAL INDEX `g_i_seller`(`seller_id`) dbpartition by hash(`seller_id`),
 UNIQUE GLOBAL INDEX `g_i_buyer` (`buyer_id`) COVERING(`seller_id`, `order_snapshot`)
  dbpartition by hash(`buyer_id`) tbpartition by hash(`buyer_id`) tbpartitions 3
) ENGINE=InnoDB DEFAULT CHARSET=utf8 dbpartition by hash(`order_id`);
```

- Specify the `g_i_seller` GSI by using FORCE INDEX in the FROM clause:

```
SELECT a.*, b.order_id
   FROM t_seller a
     JOIN t_order b FORCE INDEX(g_i_seller) ON a.seller_id = b.seller_id
   WHERE a.seller_nick="abc";
```

- Specify the `g_i_buyer` GSI by using the combination of the index name and table alias:

```
/*+TDDL:index(a, g_i_buyer)*/ SELECT * FROM t_order a WHERE a.buyer_id = 123
```

12.Functions

12.1. Functions

This topic describes the function that are supported by PolarDB-X 1.0 and some functions that are not supported by this system.

PolarDB-X 1.0 supports the date and time functions, string functions, conversion functions, aggregate functions, mathematical functions, comparison functions, bit functions, flow control functions, information functions, encryption functions, compression functions, and other functions. In addition, JSON functions and geographic information functions can be pushed down and executed.

In the WHERE clause or the UPDATE statement, PolarDB-X 1.0 does not support the following functions: LAST_INSERT_ID(), CONNECTION_ID(), CURRENT_USER(), CURRENT_USER DATABASE(), SCHEMA(), USER(), and VERSION().

PolarDB-X 1.0 does not support the following functions that are supported by MySQL 5.7:

- Full-text search functions
- XML functions
- Global transaction identifier (GTID) functions
- Enterprise encryption functions

The following functions belong to the supported function types. However, these functions are not supported.

Type	Function	Description
Date and time functions	CONVERT_TZ()	Convert from one time zone to another
	GET_FORMAT()	Return a date format string
	LOCALTIME(), LOCALTIME	Synonym for NOW()
	LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
String functions	FIND_IN_SET()	Return the index position of the first argument within the second argument
	LOAD_FILE()	Load the named file
	MATCH	Perform full-text search
	SOUNDS LIKE	Compare sounds
	BIT_AND(	Return bitwise AND
	BIT_OR()	Return bitwise OR
	BIT_XOR()	Return bitwise XOR

Aggregate functions	GROUP_CONCAT()	Return a concatenated string
	STD()	Return the population standard deviation
	STDDEV()	Return the population standard deviation
	STDDEV_POP()	Return the population standard deviation
	STDDEV_SAMP()	Return the sample standard deviation
	VAR_POP()	Return the population standard variance
	VAR_SAMP()	Return the sample variance
	VARIANCE()	Return the population standard variance
Mathematical functions	RADIANS()	Return argument converted to radians
Information functions	BENCHMARK()	Repeatedly execute an expression
	CHARSET()	Return the character set of the argument
	COERCIBILITY()	Return the collation coercibility value of the string argument
	COLLATION()	Return the collation of the string argument
	FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
	ROW_COUNT()	The number of rows updated
	ASYMMETRIC_DECRYPT()	Decrypt ciphertext using private or public key
	ASYMMETRIC_DERIVE()	Derive symmetric key from asymmetric keys
	ASYMMETRIC_ENCRYPT()	Encrypt cleartext using private or public key
	ASYMMETRIC_SIGN()	Generate signature from digest

Encryption functions and compression functions	ASYMMETRIC_VERIFY()	Verify that signature matches digest
	CREATE_ASYMMETRIC_PRIV_KEY()	Create private key
	CREATE_ASYMMETRIC_PUB_KEY()	Create public key
	CREATE_DH_PARAMETERS()	Generate shared DH secret
	CREATE_DIGEST()	Generate digest from string
	DECODE() (deprecated 5.7.2)	Decodes a string encrypted using ENCODE()
	DES_DECRYPT() (deprecated 5.7.6)	Decrypt a string
	DES_ENCRYPT() (deprecated 5.7.6)	Encrypt a string
	ENCODE() (deprecated 5.7.2)	Encode a string
	ENCRYPT() (deprecated 5.7.6)	Encrypt a string
	OLD_PASSWORD()	Return the value of the pre-4.1 implementation of PASSWORD
	PASSWORD() (deprecated 5.7.6)	Calculate and return a password string
	RANDOM_BYTES()	Return a random byte vector
	SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
	SHA2()	Calculate an SHA-2 checksum
	VALIDATE_PASSWORD_STRENGTH()	Determine strength of password
	ANY_VALUE()	Suppress ONLY_FULL_GROUP_BY value rejection
	DEFAULT()	Return the default value for a table column
	GET_LOCK()	Get a named lock
	INET_ATON()	Return the numeric value of an IP address
	INET_NTOA()	Return the IP address from a numeric value

Other functions	INET6_ATON()	Return the numeric value of an IPv6 address
	INET6_NTOA()	Return the IPv6 address from a numeric value
	IS_FREE_LOCK()	Whether the named lock is free
	IS_IPV4()	Whether argument is an IPv4 address
	IS_IPV4_COMPAT()	Whether argument is an IPv4-compatible address
	IS_IPV4_MAPPED()	Whether argument is an IPv4-mapped address
	IS_IPV6()	Whether argument is an IPv6 address
	IS_USED_LOCK()	Whether the named lock is in use; return connection identifier if true
	MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
	NAME_CONST()	Causes the column to have the given name

12.2. Date and time functions

This topic describes the date and time functions that are supported or not supported by PolarDB-X 1.0.

Supported functions

PolarDB-X 1.0 supports the following date and time functions.

Function	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time

Function	Description
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
HOUR()	Extract the hour
LAST_DAY()	Return the last day of the month for the argument
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format

Function	Description
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Functions that are not supported

PolarDB-X 1.0 does not support the following date and time functions that are supported by MySQL 5.7.

Function	Description
CONVERT_TZ()	Convert from one time zone to another
GET_FORMAT()	Return a date format string
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()

In addition, the `UNIX_TIMESTAMP()` function without parameters is not supported. We recommend that you use the `UNIX_TIMESTAMP(NOW())` function instead.

12.3. String functions

This topic describes the string functions that are supported or not supported by PolarDB-X 1.0.

Supported functions

PolarDB-X 1.0 supports the following string functions.

Function	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Return the index (position) of the first argument in the subsequent arguments
FORMAT()	Return a number formatted to specified number of decimal places

Function	Description
FROM_BASE64()	Decode to a base-64 string and return result
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Whether string matches regular expression

Function	Description
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Whether string matches regular expression
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
TO_BASE64()	Return the argument converted to a base-64 string
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for a string

Functions that are not supported

PolarDB-X 1.0 does not support the following string functions that are supported by MySQL 5.7.

Function	Description
FIND_IN_SET()	Return the index position of the first argument within the second argument
LOAD_FILE()	Load the named file
MATCH	Perform full-text search

Function	Description
SOUNDS LIKE	Compare sounds

12.4. Conversion functions

This topic describes the conversion functions that are supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following conversion functions.

Function	Description
BINARY	Cast a string to a binary string
CAST()	Cast a value as a certain type
CONVERT()	Cast a value as a certain type

The CONVERT() function supports only the CONVERT(expr USING transcoding_name) syntax. If you need to use CONVERT(expr,type), use CAST(expr AS type) instead.

12.5. Aggregate functions

This topic describes the aggregate functions. Some functions are not supported by PolarDB-X 1.0.

Supported functions

The following table describes the aggregate functions that are supported by PolarDB-X 1.0.

Function	Description
AVG()	Return the average value of the argument
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
MAX()	Return the maximum value
MIN()	Return the minimum value
SUM()	Return the sum

Unsupported functions

The following table describes the aggregate functions that are not supported by PolarDB-X 1.0. These functions are supported by databases that run on the MySQL 5.7 engine.

Function	Description
BIT_AND()	Return bitwise AND

Function	Description
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
GROUP_CONCAT()	Return a concatenated string
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

12.6. Mathematical functions

This topic describes the mathematical functions supported and not supported by Distributed Relational Database Service (PolarDB-X 1.0).

Context

Supported functions

PolarDB-X 1.0 supports the following mathematical functions.

Function	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CONV()	Convert numbers between different number bases

Function	Description
COS()	Return the cosine
COT()	Return the cotangent
CRC32()	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
MOD()	Return the remainder
PI()	Return the value of pi
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
TRUNCATE()	Truncate to specified number of decimal places

Functions that are not supported

Compared with MySQL 5.7, DRDS does not support the following mathematical functions:

Function	Description
RADIANS()	Return argument converted to radians

12.7. Comparison functions

This topic describes the comparison functions that are supported by PolarDB-X 1.0.

The following table describes the comparison functions that are supported by PolarDB-X 1.0.

Function	Description
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN()	Check whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
NOT IN()	Check whether a value is not within a set of values
STRCMP()	Compare two strings

12.8. Bit functions

This topic describes the bit function BIT_COUNT().

PolarDB-X 1.0 supports only the bit function BIT_COUNT(). The function returns the number of 1s in binary representation of an integer. The function returns NULL if the argument is NULL.

```
mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
+--------------+----------------------+
| BIT_COUNT(29) | BIT_COUNT(b'101010') |
+--------------+----------------------+
|            4 |                    3 |
+--------------+----------------------+
1 row in set (0.00 sec)
mysql> SELECT BIT_COUNT(NULL);
+-----------------+
| BIT_COUNT(NULL) |
+-----------------+
|            NULL |
+-----------------+
1 row in set (0.00 sec)
```

12.9. Flow control functions

This topic describes the flow control functions that are supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following flow control functions.

Function	Description
CASE	Case operator
IF()	If/else construct
IFNULL()	Null if/else construct
NULLIF()	Return NULL if expr1 = expr2

12.10. Information functions

Information functions are used to retrieve dynamic database information. This topic describes the information functions that are supported or not supported by PolarDB-X 1.0.

Supported functions

PolarDB-X 1.0 supports the following information functions.

Function	Description
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
CURRENT_USER(), CURRENT_USER	The authenticated user name and host name
DATABASE()	Return the default (current) database name
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
SCHEMA()	Synonym for DATABASE()
SESSION_USER()	Synonym for USER()
SYSTEM_USER()	Synonym for USER()
USER()	The user name and host name provided by the client
VERSION()	Return a string that indicates the MySQL server version

Functions that are not supported

PolarDB-X 1.0 does not support the following information functions that are supported by MySQL 5.7.

Function	Description
BENCHMARK()	Repeatedly execute an expression
CHARSET()	Return the character set of the argument

Function	Description
COERCIBILITY()	Return the collation coercibility value of the string argument
COLLATION()	Return the collation of the string argument
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
ROW_COUNT()	The number of rows updated

12.11. Encryption functions and compression functions

This topic describes the encryption functions and compression functions. Some functions are not supported by PolarDB-X 1.0.

Supported encryption functions and compression functions

The following table describes the encryption functions and compression functions that are supported by PolarDB-X 1.0.

Function	Description
AES_ENCRYPT()	Encrypt using AES
AES_DECRYPT()	Decrypt using AES
MD5()	Calculate MD5 checksum
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression

Unsupported encryption functions and compression functions

The following table describes the encryption functions and compression functions that are not supported by PolarDB-X 1.0. These functions are supported by databases that run on the MySQL 5.7 engine.

Function	Description
ASYMMETRIC_DECRYPT()	Decrypt ciphertext using private or public key
ASYMMETRIC_DERIVE()	Derive symmetric key from asymmetric keys
ASYMMETRIC_ENCRYPT()	Encrypt cleartext using private or public key
ASYMMETRIC_SIGN()	Generate signature from digest

Function	Description
ASYMMETRIC_VERIFY()	Verify that signature matches digest
CREATE_ASYMMETRIC_PRIV_KEY()	Create private key
CREATE_ASYMMETRIC_PUB_KEY()	Create public key
CREATE_DH_PARAMETERS()	Generate shared DH secret
CREATE_DIGEST()	Generate digest from string
DECODE() (deprecated 5.7.2)	Decodes a string encrypted using ENCODE()
DES_DECRYPT() (deprecated 5.7.6)	Decrypt a string
DES_ENCRYPT() (deprecated 5.7.6)	Encrypt a string
ENCODE() (deprecated 5.7.2)	Encode a string
ENCRYPT() (deprecated 5.7.6)	Encrypt a string
OLD_PASSWORD()	Return the value of the pre-4.1 implementation of PASSWORD
PASSWORD() (deprecated 5.7.6)	Calculate and return a password string
RANDOM_BYTES()	Return a random byte vector
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password

12.12. Window functions

The traditional GROUP BY function organizes data into groups and aggregates query results based on groups. In this case, GROUP BY returns only one row for each data group. However, window functions, also called online analytical processing (OLAP) functions, can return multiple rows for each data group without aggregating query results. This is different from the traditional GROUP BY function. This topic describes how to use window functions.

Prerequisites

The PolarDB-X 1.0 instance version is 5.4.8 or later.

Limits

- Window functions can be used only in SELECT statements.
- Window functions cannot be used in conjunction with the separate aggregate functions.

  In the following statement, the SUM function that does not include the OVER keyword is an aggregate function. Therefore, this statement cannot be executed.

```
SELECT SUM(NAME),COUNT() OVER(...) FROM SOME_TABLE
```

To implement the preceding query, use the following statement:

```
SELECT SUM(NAME),WIN1 FROM (SELECT NAME,COUNT() OVER(...) AS WIN1 FROM SOME_TABLE) alias
```

Syntax

```
function OVER ([[partition by column_some1] [order by column_some2] [RANGE|ROWS BETWEEN sta
rt AND end]])
```

Parameter	Description

Parameter	Description

`function`	The window function that you can specify. The following functions are supported:
	• Window functions that consist of aggregate functions and the OVER keyword:
	  ○ `SUM()`
	  ○ `COUNT()`
	  ○ `AVG()`
	  ○ `MAX()`
	  ○ `MIN()`
	• Dedicated window functions:
	  ○ `ROW_NUMBER()`
	  ○ `RANK()`
	  ○ `DESNCE_RANK()`
	  ○ `PERCENT_RANK()`
	  ○ `CUME_DIST()`
	  ○ `FIRST_VALUE()`
	  ○ `LAST_VALUE()`
	  ○ `LAG()`
	  ○ `LEAD()`
	  ○ `NTH_VALUE()`
	Note
	• When you use the `RANK()` or `DENSE_RANK()` window function, the `ORDER BY` clause cannot be omitted. For more information about dedicated window functions, see Window function descriptions.
	• Only instances whose version is 5.4.9 or later (If your instance version is earlier than 5.4.9, upgrade the version. For more information, see Upgrade the version) support the following dedicated window functions.
	  ○ `PERCENT_RANK()`
	  ○ `CUME_DIST()`
	  ○ `FIRST_VALUE()`
	  ○ `LAST_VALUE()`
	  ○ `LAG()`
	  ○ `LEAD()`
	  ○ `NTH_VALUE()`

Parameter	Description
`[partition by column_some1]`	The partition rule for the window function. This clause divides input rows into different partitions. The process is similar to the division process of the GROUP BY clause. **Note** You cannot reference complex expressions in the `PARTITION BY` clause. For example, you can reference `column_some1`, but cannot reference `column_some1 + 1`.
`[order by column_some2]`	The sorting rule for the window function. This clause defines the order in which the input rows are calculated in the window function. **Note** You cannot reference complex expressions in the `ORDER BY` clause. For example, you can reference `column_some2`, but cannot reference `column_some2 + 1`.

`[RANGE|ROWS BETWEEN start AND end]`	The window frame of the window function. You can use RANGE or ROWS to define the frame. RANGE indicates that the frame is defined by the value range for the computed column. ROWS indicates that the frame is defined by the number of rows for the computed column.
	You can use the `BETWEEN start AND end` option to specify the boundary rows in the window.
	• Valid values of `start`:
	  ○ `CURRENT ROW`: The window starts at the current row.
	  ○ `N PRECEDING`: The window starts at the preceding Nth row.
	  ○ `UNBOUNDED PRECEDING`: The window starts at the first row.
	• Valid values of `end`:
	  ○ `CURRENT ROW`: The window ends at the current row.
	  ○ `N FOLLOWING`: The window ends at the following Nth row.
	  ○ `UNBOUNDED FOLLOWING`: The window ends at the last row.

Use cases

Assume that the following raw data has been created.

```
| year | country | product    | profit |
|------|---------|------------|--------|
| 2001 | Finland | Phone      |     10 |
| 2000 | Finland | Computer   |   1500 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   1500 |
| 2000 | India   | Calculator |     75 |
| 2000 | India   | Calculator |     75 |
| 2001 | India   | Calculator |     79 |
```

- Use the following aggregate function to calculate the total profit of each country:

```
select
    country,
    sum(profit) over (partition by country) sum_profit
from test_window;
```

The following result is returned:

```
| country | sum_profit |
|---------|------------|
| India   |        229 |
| India   |        229 |
| India   |        229 |
| USA     |       1550 |
| USA     |       1550 |
| Finland |       1510 |
| Finland |       1510 |
```

- Use the following dedicated window function to group data by country and rank the products of each country by profit in ascending order:

```
select
    'year',
    country,
    product,
    profit,
    rank() over (partition by country order by profit) as rank
from test_window;
```

The following result is returned:

```
| year | country | product    | profit | rank |
|------|---------|------------|--------|------|
| 2001 | Finland | Phone      |     10 |    1 |
| 2000 | Finland | Computer   |   1500 |    2 |
| 2001 | USA     | Calculator |     50 |    1 |
| 2001 | USA     | Computer   |   1500 |    2 |
| 2000 | India   | Calculator |     75 |    1 |
| 2000 | India   | Calculator |     75 |    1 |
| 2001 | India   | Calculator |     79 |    3 |
```

- Execute the following statement that contains the ROWS option to calculate a cumulative sum of profits for each row in the current window:

```
select
    'year',
    country,
    profit,
    sum(profit) over (partition by country order by 'year' ROWS BETWEEN UNBOUNDED PRECEDI
NG and CURRENT ROW) as sum_win
from test_window;
```

The following result is returned:

```
+------+---------+--------+-------------+
| year | country | profit |   sum_win   |
+------+---------+--------+-------------+
| 2001 | USA     |     50 |          50 |
| 2001 | USA     |   1500 |        1550 |
| 2000 | India   |     75 |          75 |
| 2000 | India   |     75 |         150 |
| 2001 | India   |     79 |         229 |
| 2000 | Finland |   1500 |        1500 |
| 2001 | Finland |     10 |        1510 |
```

12.13. Other functions

This topic describes other functions that are supported by PolarDB-X 1.0.

The following table describes other functions that are supported by PolarDB-X 1.0.

Function	Description
RAND()	Return a random floating-point value
RELEASE_ALL_LOCKS()	Releases all current named locks
RELEASE_LOCK()	Releases the named lock
SLEEP()	Sleep for a number of seconds
UUID()	Return a Universal Unique Identifier (UUID)
UUID_SHORT()	Return an integer-valued universal identifier
VALUES()	Defines the values to be used during an INSERT
ANY_VALUE()	Suppress ONLY_FULL_GROUP_BY value rejection
DEFAULT()	Return the default value for a table column
GET_LOCK()	Get a named lock
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value

Function	Description
INET6_ATON()	Return the numeric value of an IPv6 address
INET6_NTOA()	Return the IPv6 address from a numeric value
IS_FREE_LOCK()	Whether the named lock is free
IS_IPV4()	Whether argument is an IPv4 address
IS_IPV4_COMPAT()	Whether argument is an IPv4-compatible address
IS_IPV4_MAPPED()	Whether argument is an IPv4-mapped address
IS_IPV6()	Whether argument is an IPv6 address
IS_USED_LOCK()	Whether the named lock is in use; return connection identifier if true
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
NAME_CONST()	Causes the column to have the given name

12.14. GROUPING SETS, ROLLUP, and CUBE extensions

In relational databases, you must use multiple `SELECT and UNION` statements to group results based on multiple groups of dimensions. PolarDB-X 1.0 provides GROUPING SETS, ROLLUP, and CUBE extensions that allow you to group results based on multiple groups of dimensions. In addition, PolarDB-X 1.0 allows you to use the GROUPING and GROUPING_ID functions in a SELECT statement or a HAVING clause. This helps to explain the results of the preceding extensions. This topic describes the relevant syntax and examples.

Prerequisites

The PolarDB-X 1.0 instance version is 5.4.10 or later.

Considerations

- The syntax of all the GROUP BY extensions in this topic does not allow SQL queries to be pushed down to the `LogicalView` operators for execution. For more information about SQL query pushdown, see SQL rewrite and pushdown.
- The following test data information is used in the examples of this topic:

  Execute the following statement to create a table named `requests`:

```
CREATE TABLE requests (
  `id` int(10) UNSIGNED NOT NULL,
  `os` varchar(20) DEFAULT NULL,
  `device` varchar(20) DEFAULT NULL,
  `city` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE = InnoDB DEFAULT CHARSET = utf8 dbpartition BY hash(`id`) tbpartition BY hash(`i
d`);
```

Execute the following statement to insert the required test data to the `requests` table:

```
INSERT INTO requests (id, os, device, city) VALUES
(1, 'windows', 'PC', 'Beijing'),
(2, 'windows', 'PC', 'Shijiazhuang'),
(3, 'linux', 'Phone', 'Beijing'),
(4, 'windows', 'PC', 'Beijing'),
(5, 'ios', 'Phone', 'Shijiazhuang'),
(6, 'linux', 'PC', 'Beijing'),
(7, 'windows', 'Phone', 'Shijiazhuang');
```

GROUPING SETS extension

- **Overview**

  GROUPING SETS is an extension of the GROUP BY clause and can generate a result set. The result set is a concatenation of multiple result sets based on different groups. The result returned by the GROUPING SETS extension is similar to that of the UNION ALL operator. However, the UNION ALL operator and the GROUPING SETS expansion do not remove duplicate rows from the combined result sets.

- **Syntax**

```
GROUPING SETS (
  { expr_1 | ( expr_1a [, expr_1b ] ...) |
    ROLLUP ( expr_list ) | CUBE ( expr_list )
  } [, ...] )
```

  Note A GROUPING SETS extension can contain a combination of one or more comma-separated expressions, such as `expr_1` or `(expr_1a [, expr_1b ] ...)`,and lists of expressions enclosed within parentheses (), such as `( expr_list )`. In the syntax:

  - Each expression can be used to determine how the result set is grouped.
  - You can nest a ROLLUP or CUBE extension in a GROUPING SETS extension.

- **Examples**

- You can group the data that you want to query by using a GROUPING SETS extension. The following code block shows the relevant syntax:

```
select os,device, city ,count(*)
from requests
group by grouping sets((os, device), (city), ());
The preceding statement is equivalent to the following statement:
select os, device, NULL, count(*)
from requests group by os, device
union all
select NULL, NULL, NULL, count(*)
from requests
union all
select null, null, city, count(*)
from requests group by city;
```

The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| windows | PC     | NULL         |        3 |
| linux   | PC     | NULL         |        1 |
| linux   | Phone  | NULL         |        1 |
| windows | Phone  | NULL         |        1 |
| ios     | Phone  | NULL         |        1 |
| NULL    | NULL   | Shijiazhuang |        3 |
| NULL    | NULL   | Beijing      |        4 |
| NULL    | NULL   | NULL         |        7 |
+---------+--------+--------------+----------+
```

Note If an expression is not used in a grouping set, NULL is used as a placeholder for the expression. This facilitates operations on the result set that is not used in the grouping set. For example, the result set is the rows where the values of the `city` column are NULL in the returned result.

- You can group data by nesting a ROLLUP extension in a GROUPING SETS extension. The following code block shows the relevant syntax:

```
select os,device, city ,count(*) from requests
group by grouping sets((city), ROLLUP(os, device));
The preceding statement is equivalent to the following statement:
select os,device, city ,count(*) from requests
group by grouping sets((city), (os), (os, device), ());
```

The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| NULL    | NULL   | Shijiazhuang |        3 |
| NULL    | NULL   | Beijing      |        4 |
| windows | PC     | NULL         |        3 |
| linux   | PC     | NULL         |        1 |
| ios     | Phone  | NULL         |        1 |
| linux   | Phone  | NULL         |        1 |
| windows | Phone  | NULL         |        1 |
| windows | NULL   | NULL         |        4 |
| linux   | NULL   | NULL         |        2 |
| ios     | NULL   | NULL         |        1 |
| NULL    | NULL   | NULL         |        7 |
+---------+--------+--------------+----------+
```

- You can group data by nesting a CUBE extension in a GROUPING SETS extension. The following code block shows the relevant syntax:

```
select os,device, city ,count(*) from requests
group by grouping sets((city), CUBE(os, device));
The preceding statement is equivalent to the following statement:
select os,device, city ,count(*) from requests
group by grouping sets((city), (os), (os, device), (), (device));
```

The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| NULL    | NULL   | Beijing      |        4 |
| NULL    | NULL   | Shijiazhuang |        3 |
| windows | PC     | NULL         |        3 |
| ios     | Phone  | NULL         |        1 |
| linux   | Phone  | NULL         |        1 |
| windows | Phone  | NULL         |        1 |
| linux   | PC     | NULL         |        1 |
| windows | NULL   | NULL         |        4 |
| ios     | NULL   | NULL         |        1 |
| linux   | NULL   | NULL         |        2 |
| NULL    | PC     | NULL         |        4 |
| NULL    | Phone  | NULL         |        3 |
| NULL    | NULL   | NULL         |        7 |
+---------+--------+--------------+----------+
```

- You can combine the GROUP BY clause and the CUBE, and GROUPING SETS extensions to generate grouping sets, as shown in the following example:

```
select os,device, city, count(*)
from requests
group by os, cube(os,device), grouping sets(city);
The preceding statement is equivalent to the following statement:
select os,device, city, count(*)
from requests
group by grouping sets((os,device,city),(os,city),(os,device,city));
```

The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| linux   | Phone  | Beijing      |        1 |
| windows | Phone  | Shijiazhuang |        1 |
| windows | PC     | Shijiazhuang |        1 |
| linux   | PC     | Beijing      |        1 |
| windows | PC     | Beijing      |        2 |
| ios     | Phone  | Shijiazhuang |        1 |
| linux   | NULL   | Beijing      |        2 |
| windows | NULL   | Shijiazhuang |        2 |
| windows | NULL   | Beijing      |        2 |
| ios     | NULL   | Shijiazhuang |        1 |
+---------+--------+--------------+----------+
```

ROLLUP extension

- **Overview**

  A ROLLUP extension generates a hierarchical set of groups. In this set, subtotals for each hierarchical group and a grand total are available. The order of the hierarchy is determined by the order of the expressions that are specified in the ROLLUP expression list. The top of the hierarchy is the leftmost item in the list. Each successive item that proceeds to the right side moves down the hierarchy. The rightmost item is at the lowest level.

- **Syntax**

```
ROLLUP ( { expr_1 | ( expr_1a [, expr_1b ] ...) }
  [, expr_2 | ( expr_2a [, expr_2b ] ...) ] ...)
```

> **Note**
>
> - Each expression is used to determine how the result set is grouped. If the expressions are enclosed in parentheses (), such as `( expr_1a, expr_1b, ...)`, the combination of values returned by `expr_1a` and `expr_1b` defines a single grouping level of the hierarchy.
> - For the first item in the list, such as `expr_1` or the combination of `( expr_1a, expr_1b, ...)`, PolarDB-X 1.0a subtotal is returned for each unique value of the first item. For the second item in the list, such as `expr_2` or the combination of `( expr_2a, expr_2b, ...)`, PolarDB-X 1.0a subtotal is returned for each unique value of each group in the second item. Similar rules are used in each grouping level of the first item and other items. Finally, PolarDB-X 1.0a grand total is returned for the entire result set.
> - For the subtotal rows, NULL is returned for the items across which the subtotal is taken.

- **Examples**
  - ROLLUP is used to aggregate `(os, device, city)` in a hierarchical manner to generate grouping sets. The following code block shows the relevant syntax:

```
select os,device, city, count(*)
from requests
group by rollup (os, device, city);
The preceding statement is equivalent to the following statement:
select os,device, city, count(*)
from requests
group by os, device, city with rollup;
The first statement is also equivalent to the following statement:
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(os),());
```

    The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| windows | PC     | Beijing      |        2 |
| ios     | Phone  | Shijiazhuang |        1 |
| windows | PC     | Shijiazhuang |        1 |
| linux   | PC     | Beijing      |        1 |
| linux   | Phone  | Beijing      |        1 |
| windows | Phone  | Shijiazhuang |        1 |
| windows | PC     | NULL         |        3 |
| ios     | Phone  | NULL         |        1 |
| linux   | PC     | NULL         |        1 |
| linux   | Phone  | NULL         |        1 |
| windows | Phone  | NULL         |        1 |
| windows | NULL   | NULL         |        4 |
| ios     | NULL   | NULL         |        1 |
| linux   | NULL   | NULL         |        2 |
| NULL    | NULL   | NULL         |        7 |
+---------+--------+--------------+----------+
```

- ROLLUP is used to aggregate `os, (os,device), and city` in a hierarchical manner to generate grouping sets. The following code block shows the relevant syntax:

```
select os,device, city, count(*)
from requests
group by rollup (os, (os,device), city);
The preceding statement is equivalent to the following statement:
select os,device, city, count(*)
from requests
group by os, (os,device), city with rollup;
The first statement is also equivalent to the following statement:
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(os),());
```

The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| windows | PC     | Beijing      |        2 |
| windows | PC     | Shijiazhuang |        1 |
| linux   | PC     | Beijing      |        1 |
| linux   | Phone  | Beijing      |        1 |
| windows | Phone  | Shijiazhuang |        1 |
| ios     | Phone  | Shijiazhuang |        1 |
| windows | PC     | NULL         |        3 |
| linux   | PC     | NULL         |        1 |
| linux   | Phone  | NULL         |        1 |
| windows | Phone  | NULL         |        1 |
| ios     | Phone  | NULL         |        1 |
| windows | NULL   | NULL         |        4 |
| linux   | NULL   | NULL         |        2 |
| ios     | NULL   | NULL         |        1 |
| NULL    | NULL   | NULL         |        7 |
+---------+--------+--------------+----------+
```

CUBE extension

- **Overview**

  A CUBE extension is similar to a ROLLUP extension. A ROLLUP extension generates groupings and results in a hierarchy based on a left-to-right listing of items in the ROLLUP expression list. However, a CUBE extension generates groupings and subtotals based on each permutation of all the items in the CUBE expression list. Therefore, a CUBE extension returns more rows in the generated result set than a ROLLUP extension that is performed on the same expression list.

- **Syntax**

```
CUBE ( { expr_1 | ( expr_1a [, expr_1b ] ...) }
  [, expr_2 | ( expr_2a [, expr_2b ] ...) ] ...)
```

> **Note**
>
> - Each expression is used to determine how the result set is grouped. If the expressions are enclosed within parentheses (), such as `( expr_1a, expr_1b, ...)`, the combination of values that are returned by `expr_1a` and `expr_1b` defines a single group.
> - For the first item in the list, such as `expr_1` or the combination of `( expr_1a, expr_1b, ...)`, PolarDB-X 1.0a subtotal is returned for each unique value of the first item. For the second item in the list, such as `expr_2` or the combination of `( expr_2a, expr_2b, ...)`, PolarDB-X 1.0a subtotal is returned for each unique value of the second item. A subtotal is also returned for each unique combination of the first item and the second item. If a third item exists, PolarDB-X 1.0a subtotal is returned for each unique value of the third item, each unique combination of the third and first items, each unique combination of the third and second items, and each unique combination of the third, second, and first items. Finally, a grand total is returned for the entire result set.
> - For the subtotal rows, NULL is returned for the items across which the subtotal is taken.

- **Examples**
  - CUBE lists all the possible combinations of `(os, device, city)` columns as grouping sets. The following code block shows the relevant syntax:

```
select os,device, city, count(*)
from requests
group by cube (os, device, city);
The preceding statement is equivalent to the following statement:
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(os, city),(device,city),(os),(
device),(city),());
```

    The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| linux   | Phone  | Beijing      |        1 |
| windows | Phone  | Shijiazhuang |        1 |
| windows | PC     | Beijing      |        2 |
| ios     | Phone  | Shijiazhuang |        1 |
| windows | PC     | Shijiazhuang |        1 |
| linux   | PC     | Beijing      |        1 |
| linux   | Phone  | NULL         |        1 |
| windows | Phone  | NULL         |        1 |
| windows | PC     | NULL         |        3 |
| ios     | Phone  | NULL         |        1 |
| linux   | PC     | NULL         |        1 |
| linux   | NULL   | Beijing      |        2 |
| windows | NULL   | Shijiazhuang |        2 |
| windows | NULL   | Beijing      |        2 |
| ios     | NULL   | Shijiazhuang |        1 |
| linux   | NULL   | NULL         |        2 |
| windows | NULL   | NULL         |        4 |
| ios     | NULL   | NULL         |        1 |
| NULL    | Phone  | Beijing      |        1 |
| NULL    | Phone  | Shijiazhuang |        2 |
| NULL    | PC     | Beijing      |        3 |
| NULL    | PC     | Shijiazhuang |        1 |
| NULL    | Phone  | NULL         |        3 |
| NULL    | PC     | NULL         |        4 |
| NULL    | NULL   | Beijing      |        4 |
| NULL    | NULL   | Shijiazhuang |        3 |
| NULL    | NULL   | NULL         |        7 |
+---------+--------+--------------+----------+
```

- CUBE lists all the possible combinations of `(os, device),(device, city)` columns as grouping sets. The following code block shows the relevant syntax:

```
select os,device, city, count(*)
from requests
group by cube ((os, device), (device, city));
The preceding statement is equivalent to the following statement:
select os,device, city, count(*)
from requests
group by grouping sets ((os, device, city),(os, device),(device,city),());
```

The following result is returned:

```
+---------+--------+--------------+----------+
| os      | device | city         | count(*) |
+---------+--------+--------------+----------+
| linux   | Phone  | Beijing      |        1 |
| windows | Phone  | Shijiazhuang |        1 |
| windows | PC     | Beijing      |        2 |
| windows | PC     | Shijiazhuang |        1 |
| linux   | PC     | Beijing      |        1 |
| ios     | Phone  | Shijiazhuang |        1 |
| linux   | Phone  | NULL         |        1 |
| windows | Phone  | NULL         |        1 |
| windows | PC     | NULL         |        3 |
| linux   | PC     | NULL         |        1 |
| ios     | Phone  | NULL         |        1 |
| NULL    | Phone  | Beijing      |        1 |
| NULL    | Phone  | Shijiazhuang |        2 |
| NULL    | PC     | Beijing      |        3 |
| NULL    | PC     | Shijiazhuang |        1 |
| NULL    | NULL   | NULL         |        7 |
+---------+--------+--------------+----------+
```

GROUPING and GROUPING_ID functions

- **Overview**
  - **GROUPING function**

    When you use the GROUPING SETS, ROLLUP, or CUBE extensions in the GROUP BY clause, NULL is used as a placeholder in a return value of the GROUPING SETS extension. As a result, the placeholder NULL cannot be distinguished from the value NULL. You can use the GROUPING function provided by PolarDB-X 1.0 to solve this problem.

    The GROUPING function allows you to use a column name as a parameter. If the corresponding rows are aggregated based on the column, 0 is returned in the result. In this case, NULL is a value. If the corresponding rows are not aggregated based on the column, 1 is returned. In this case, NULL is a placeholder in a return value of the GROUPING SETS extension.

- **GROUPING_ID function**

  The GROUPING_ID function simplifies the process of implementing the GROUPING function. The GROUPING_ID function is used to determine the subtotal level of a row in the result set from a ROLLBACK, CUBE, or GROUPING SETS extension. The GROUPING function uses only one column expression and returns a value to indicate whether a row is a subtotal for all the values of the specified column. Therefore, multiple GROUPING functions may be required to interpret the level of subtotals for queries that have multiple grouping columns. The GROUPING_ID function supports one or more column expressions that have been used in the ROLLBACK, CUBE, or GROUPING SETS extensions and returns a single integer. This integer indicates the column on which a subtotal has been aggregated.

- **Syntax**
  - **GROUPING function**

```
SELECT [ expr ...,] GROUPING( col_expr ) [, expr ] ...
FROM ...
GROUP BY { ROLLUP | CUBE | GROUPING SETS }( [...,] col_expr
  [, ...] ) [, ...]
```

> **Note** The GROUPING function uses a single parameter. This parameter must be an expression of a dimension column that is specified in the expression list of a ROLLUP, CUBE, or GROUPING SETS extension of the GROUP BY clause.

  - **GROUPING_ID function**

```
SELECT [ expr ...,]
  GROUPING_ID( col_expr_1 [, col_expr_2 ] ... )
  [, expr ] ...
FROM ...
GROUP BY { ROLLUP | CUBE | GROUPING SETS }( [...,] col_expr_1
  [, col_expr_2 ] [, ...] ) [, ...]
```

- **Examples**

  The GROUPING_ID function uses multiple column names as parameters, and converts the grouping results of the parameter columns into integers by using the bitmap algorithm. The following code block shows the relevant syntax:

```
select a,b,c,count(*),
grouping(a) ga, grouping(b) gb, grouping(c) gc, grouping_id(a,b,c) groupingid
from (select 1 as a ,2 as b,3 as c)
group by cube(a,b,c);
```

  The following result is returned:

```
+------+------+------+----------+------+------+------+------------+
| a    | b    | c    | count(*) | ga   | gb   | gc   | groupingid |
+------+------+------+----------+------+------+------+------------+
|    1 |    2 |    3 |        1 |    0 |    0 |    0 |          0 |
|    1 |    2 | NULL |        1 |    0 |    0 |    1 |          1 |
|    1 | NULL |    3 |        1 |    0 |    1 |    0 |          2 |
|    1 | NULL | NULL |        1 |    0 |    1 |    1 |          3 |
| NULL |    2 |    3 |        1 |    1 |    0 |    0 |          4 |
| NULL |    2 | NULL |        1 |    1 |    0 |    1 |          5 |
| NULL | NULL |    3 |        1 |    1 |    1 |    0 |          6 |
| NULL | NULL | NULL |        1 |    1 |    1 |    1 |          7 |
+------+------+------+----------+------+------+------+------------+
```

13.Operator

13.1. Logical operators

This topic describes the logical operators supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following logical operators.

Operator	Description
AND, &&	Logical AND
NOT, !	Negates value
||, OR	Logical OR
XOR	Logical XOR

13.2. Arithmetic operators

This topic describes the arithmetic operators supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following arithmetic operators.

Operator	Description
DIV	Integer division
/	Division operator
-	Minus operator
%, MOD	Modulo operator
+	Addition operator
*	Multiplication operator
-	Change the sign of the argument

13.3. Comparison operators

This topic describes the comparison operators supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following comparison operators.

Operator	Description
BETWEEN ... AND ...	Check whether a value is within a range of values

Operator	Description
=	Equal operator
<=>	NULL-safe equal to operator
>	Greater than operator
>=	Greater than or equal operator
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT LIKE	Negation of simple pattern matching

13.4. Bitwise operators

This topic describes the bitwise operators supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following bitwise operators.

Operator	Description
&	Bitwise AND
~	Bitwise inversion
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift

13.5. Assignment operators

This topic describes the assignment operators that are supported by PolarDB-X 1.0 and the assignment operators that are not supported by DRDS.

PolarDB-X 1.0 supports the = assignment operator. This operator is generally used in the SET clause of UPDATE statements.

PolarDB-X 1.0 does not support the := assignment operator.

13.6. Operator precedence

This topic describes the precedence of operators supported by PolarDB-X 1.0.

The following table describes the precedence of operators that are supported by PolarDB-X 1.0. The operators are listed by precedence in descending order.

Precedence	Operator
15	!
14	- (unary minus) and ~
13	^
12	*, /, %, and MOD
11	+ and -
10	<<,>>
9	&
8	|
7	= (equality operator for comparison), <=>, >, >=, <, <=, <>, !=, IS, LIKE, REGEXP, and IN
6	BETWEEN
5	NOT
4	AND, &&
3	XOR
2	OR, ||
1	= (assignment operator)

Compare the precedence of the IN and NOT IN operators and the = comparison operator

Execute the following SQL statements on a database that runs MySQL 5.7.19:

```
mysql> select binary 'a' = 'a' in (1, 2, 3);
+------------------------------+
| binary 'a' = 'a' in (1, 2, 3) |
+------------------------------+
|                            1 |
+------------------------------+
1 row in set, 1 warning (0.01 sec)
mysql> show warnings;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'a' |
+---------+------+---------------------------------------+
1 row in set (0.00 sec)
mysql> select 1 in (1, 2, 3) = 'a';
+----------------------+
| 1 in (1, 2, 3) = 'a' |
+----------------------+
|                    0 |
+----------------------+
1 row in set, 1 warning (0.00 sec)
mysql> show warnings;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'a' |
+---------+------+---------------------------------------+
1 row in set (0.00 sec)
```

This example shows that in MySQL, the IN and NOT IN operators have a higher precedence than the = comparison operator.PolarDB-X 1.0 strictly follows the precedence described in the preceding table. If two or more operators that have the same precedence are used in one SQL statement, the operators are evaluated from left to right.

14.Data types

14.1. Data types

This topic describes the data types supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports four major categories of data types:

- Numeric
- String
- Date and time
- JSON

Spatial data types are not supported.

For more information about data types, see Data types in the Reference Manual of MySQL.

14.2. Numeric data types

This topic describes the numeric data types supported by PolarDB-X 1.0.

The numeric data types can be classified into two categories by precision:

- Exact numeric data types
  - Integer data types: TINYINT, SAMLLINT, MEDIUMINT, INTEGER, and BIGINT
  - Fixed-point data types: DECIMAL and NUMERIC
- Approximate numeric data types: FLOAT, REAL, and DOUBLE PRECISION

The supported data types are consistent with those of MySQL. For more information, see Numeric data types in the Reference Manual of MySQL.

14.3. String data types

This topic describes the string data types supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following string data types:

- CHAR and VARCHAR
- BINARY and VARBINARY
- BLOB and TEXT
- ENUM
- SET

For more information, see String data types in the Reference Manual of MySQL.

14.4. Collation types

A character set is a combination of a set of symbols and encoding methods. A collation is the rules for sorting characters in a character set. This topic summarizes the collation types that PolarDB-X 1.0 supports.

> **Note** For more information about the collation types, see Collations.

Character set	collation
utf8	utf8_general_ci
	utf8_bin
	utf8_unicode_ci
utf8mb4	utf8mb4_general_ci
	utf8mb4_bin
	utf8mb4_unicode_ci
utf16	utf16_general_ci
	utf16_bin
	utf16_unicode_ci
ascii	ascii_general_ci
	ascii_bin
binary	binary
latin1	latin1_swedish_ci
	latin1_german1_ci
	latin1_danish_ci
	latin1_bin
	latin1_general_ci
	latin1_general_cs
	latin1_spanish_ci
gbk	gbk_chinese_ci
	gbk_bin

14.5. Date and time data types

This topic describes the date and time data types supported by PolarDB-X 1.0.

PolarDB-X 1.0 supports the following date and time data types:

- DATE

- DATETIME
- TIMESTAMP
- TIME
- YEAR

> **Note** The value range for the TIME data type in MySQL is different from that in PolarDB-X 1.0. In PolarDB-X 1.0, the value range for the TIME data type is '00:00:00' to '23:59:59'.

For more information, see Date and time data types in the Reference Manual of MySQL.

15.Practical SQL statements

15.1. TRACE

This topic describes how to use the TRACE statement.

You can execute the TRACE statement to view the execution result of an SQL statement. You must use the TRACE <SQL> statement and the SHOW TRACE statement together.

> **Note** The difference between the TRACE <SQL> statement and the EXPLAIN <SQL> statement is that the TRACE <SQL> statement is executed.

Examples

Use the TRACE statement to view the execution result of the select 1 statement.

```
mysql> trace select 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.03 sec)
mysql> show trace;
+------+----------+----------------+-----------------------------------+---------------+---
----------------------+------+-----------+--------+
| ID   | TYPE     | GROUP_NAME     | DBKEY_NAME                        | TIME_COST(MS) | CO
NNECTION_TIME_COST(MS) | ROWS | STATEMENT | PARAMS |
+------+----------+----------------+-----------------------------------+---------------+---
----------------------+------+-----------+--------+
|    0 | Optimize | DRDS           | DRDS                              | 3             | 0.
00                     |    0 | select 1  | NULL   |
|    1 | Query    | TDDL5_00_GROUP | db218249098_sqa_zmf_tddl5_00_3309 | 7             | 0.
15                     |    1 | select 1  | NULL   |
+------+----------+----------------+-----------------------------------+---------------+---
----------------------+------+-----------+--------+
2 rows in set (0.01 sec)
```

15.2. Cross-schema queries

In most cases, multiple schemas are used in a Distributed Relational Database Service (PolarDB-X 1.0) instance. PolarDB-X 1.0 allows you to execute SQL statements to perform cross-schema queries. The results are similar to those of cross-schema queries in MySQL.

Note

- To use the cross-schema query syntax, you must prefix the destination TableName with the corresponding SchemaName in your SQL statement. For example, if the TableName is `xxx_tbl` and the corresponding SchemaName is yyy_db, you must use `yyy_db` . `xxx_tbl` to specify the schema

to which the `xxx_tbl` table belongs. The cross-schema query syntax in PolarDB-X 1.0 is fully compatible with that in MySQL.

- PolarDB-X 1.0 does not support cross-schema queries that contain the following statements: CREATE SEQUENCE, ALTER SEQUENCE, and DROP SEQUENCE.
- The version of your PolarDB-X 1.0 instance must be V5.3.8-15517870 or later.
- Before you perform a cross-schema query, you must be granted the required permissions on the related schemas. For more information about the syntax for granting permissions, see Manage accounts and permissions.

Terms

- Schema: a database in a PolarDB-X 1.0 instance. Horizontal splitting may or may not be performed on the database.
- SchemaName: the name of a database in a PolarDB-X 1.0 instance. The name is unique within the instance.
- Table: a table in a PolarDB-X 1.0 database. Horizontal splitting may or may not be performed on the database.
- TableName: the name of a table in a PolarDB-X 1.0 database. The name is unique within the database.

Examples

If you have created three different schemas in a PolarDB-X 1.0 instance, each of the schemas contains one table and each of the tables corresponds to one sequence, as shown in the following table.

SchemaName	TableName	Sequence
`new_db`	`new_tbl`	`AUTO_SEQ_new_tbl`
`trade_db`	`trade_tbl`	`AUTO_SEQ_trade_tbl`
`user_db`	`user_tbl`	`AUTO_SEQ_user_tbl`

The SchemaName that you use to log on to the PolarDB-X 1.0 instance is `trade_db`. You can execute the following SQL statements to perform cross-schema queries:

- Execute the SELECT statement to perform cross-schema queries

  To perform an aggregate query across the `trade_tbl` schema and the `user_tbl` schema, you can execute the following SQL statement:

  ```
  SELECT COUNT(DISTINCT u.user_id)
  FROM `trade_tbl` AS t
  INNER JOIN `user_db`.`user_tbl` AS u ON t.user_id=u.user_id
  WHERE u.user_id >= 10000
  GROUP BY t.title
  ```

- Execute the INSERT statement to perform cross-schema queries

  To insert data to the `new_tbl` table in the `new_db` schema, you can execute the following SQL statement:

  ```
  INSERT INTO `new_db`.`new_tbl` (user_id, title) VALUES ( null, 'test' );
  ```

- Use a distributed transaction to perform cross-schema queries

  To update or delete the `new_tbl` table and the `user_tbl` table in a distributed transaction and commit the operations by using one request, you can execute the following SQL statements:

```
SET AUTOCOMMIT=off;
SET drds_transaction_policy = 'XA';
UPDATE `new_db`.`new_tbl` SET name='abc' WHERE use_id=1;
DELETE FROM `user_db`.`user_tbl` WHERE user_id=2;
COMMIT;
```

- Use sequences to perform cross-schema queries

  To explicitly use sequences to perform cross-schema INSERT operations, you must explicitly prefix the sequence name with the SchemaName. For example, change `xxx_seq` to `yyy_db`.`xxx_seq`.

```
/* This SQL statement uses the `AUTO_SEQ_new_tbl` table in the `new_db` schema as a seque
nce to insert data.*/
INSERT INTO `new_db`.`new_tbl` (id, name) values ( null, 'test_seq' );
```

```
/* This SQL statement uses the `AUTO_SEQ_new_tbl` table in the `new_db` schema as a seque
nce to insert data. In the sequence, the SchemaName is specified.*/
INSERT INTO `new_db`.`new_tbl` (id, name) values ( `new_db`.AUTO_SEQ_new_tbl.nextval, 'te
st_seq' );
```

- Execute the SHOW CREATE TABLE statement to perform cross-schema queries

  To query the data of another schema such as `new_db` in the current schema, you can execute the following SQL statement:

```
SHOW CREATE TABLE `new_db`.`new_tbl`;
```

SQL statements that support cross-schema queries

- SELECT
- INSERT
- REPLACE
- UPDATE
- DELETE
- Sequence
- DAL
- USE

15.3. Multiple statements

This topic describes the multiple statements feature supported by PolarDB-X 1.0).

PolarDB-X 1.0 allows you to specify multiple statements in one statement string. The statements must be separated with semicolons (;).

```
mysql> SELECT * FROM t1; SELECT * FROM t2; SELECT NOW().
```

> **Note**
>
> - Before you execute the preceding statement string, use the --delimiter parameter on the MySQL client to change the delimiter for SQL statements to a period (.) on the MySQL client. This prevents the client from splitting the SQL request based on semicolons (;).
> - When PolarDB-X 1.0 executes the preceding SQL statement string, it splits the SQL statements based on semicolons (;) and then execute the statements in sequence.

15.4. EXPLAIN and execution plans

Execution Plans

Similar to most database systems, PolarDB-X 1.0 uses an optimizer to generate an execution plan when it processes an SQL statement. This execution plan has a tree structure of relational operators, which reflects how PolarDB-X 1.0 executes the SQL statement. The difference is that PolarDB-X 1.0 does not store data but pushes computations down to each ApsaraDB RDS for MySQL database for execution while the network I/O overheads is considered in a distributed environment. In this way, the efficiency of SQL statement execution is improved. You can execute the EXPLAIN statement to view an SQL execution plan.This topic describes the meanings of the operators used in a PolarDB-X 1.0 execution plan so that you can understand the SQL execution process by using the execution plan. This helps you optimize SQL statements. The examples in this topic are based on the following table structure:

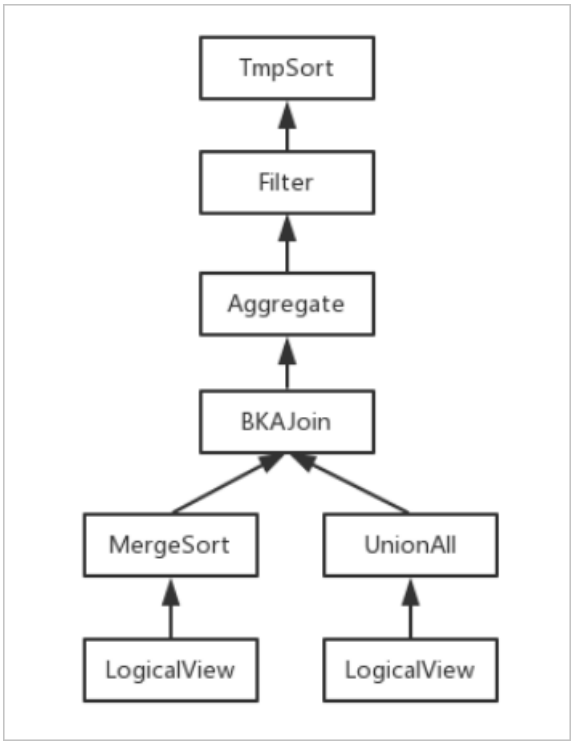
```
CREATE TABLE `sbtest1` (
  `id`  INT(10) UNSIGNED NOT NULL,
  `k`   INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `c`   CHAR(120)        NOT NULL DEFAULT '',
  `pad` CHAR(60)         NOT NULL DEFAULT '',
  KEY `xid` (`id`),
  KEY `k_1` (`k`)
) dbpartition BY HASH (`id`) tbpartition BY HASH (`id`) tbpartitions 4
```

The following example helps you understand the tree structure of an execution plan in PolarDB-X 1.0:

```
mysql> explain select a.k, count(*) cnt from sbtest1 a, sbtest1 b where a.id = b.k and a.id
> 1000 group by k having cnt > 1300 order by cnt limit 5, 10;
+-------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| TmpSort(sort="cnt ASC", offset=?2, fetch=?3)
|
|   Filter(condition="cnt > ?1")
|
|     Aggregate(group="k", cnt="COUNT()")
|
|       BKAJoin(id="id", k="k", c="c", pad="pad", id0="id0", k0="k0", c0="c0", pad0="pad0",
condition="id = k", type="inner")                        |
|         MergeSort(sort="k ASC")
|
|           LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT
* FROM `sbtest1` WHERE (`id` > ?) ORDER BY `k`")         |
|         UnionAll(concurrent=true)
|
|           LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT
* FROM `sbtest1` WHERE ((`k` > ?) AND (`k` IN ('?')))") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------
-----------------------------------------------------------+
9 rows in set (0.01 sec)
```

As shown in the preceding example, the overall results of a PolarDB-X 1.0 EXPLAIN statement are divided into two parts: the execution plan and other information.

- Execution plan: The execution plan represents the parent-child relationships between operators in indent form. In this example, the Filter is a child operator of TmpSort and a parent operator of Aggregate. From the perspective of execution, each operator pulls data from its child operators, processes the pulled data, and then exports the processed data to its parent operator. To better understand the preceding execution plan, we convert the preceding execution plan into a tree structure:



- Other information: In addition to the execution plan, other information is included in the EXPLAIN results. In this example, only `HitCache` is included. PolarDB-X 1.0 enables the PlanCache function by default. `HitCache` indicates whether the current SQL statement hits PlanCache.After PlanCache is enabled, PolarDB-X 1.0 parameterizes the SQL statement by replacing most constants with a question mark (`?`), and constructing a parameter list. For example, in the execution plan, LogicalView's SQL has a question mark (`?`), and certain operators may have some characters like `?2` . The `2` here indicates the subscript of operators in the parameter list. This will be further elaborated with specific examples later.

EXPLAIN syntax

The EXPLAIN statement is used to view the execution plan of an SQL statement. The following sample code shows the syntax:

```
EXPLAIN
{LOGICALVIEW | LOGIC | SIMPLE | DETAIL | EXECUTE | PHYSICAL | OPTIMIZER | SHARDING
 | COST | ANALYZE | BASELINE | JSON_PLAN | ADVISOR}
 {SELECT statement | DELETE statement | INSERT statement | REPLACE statement| UPDATE statem
ent}
```

Introduction to operators

LogicalView

LogicalView pulls data from the underlying data source. From the perspective of database, naming with `TableScan` is more conventional. However, given that PolarDB-X 1.0 itself does not store data but instead obtains data from the underlying data source by using SQL statements, this operator is more like a 'view' as it records the pushed down SQL statement and data source information. The SQL statements in this 'view' are pushed down by an optimizer. This may include multiple operations such as projection, filtering, aggregation, sorting, joining, and subqueries.

The following example describes the output and meanings of LogicalView in the EXPLAIN statement:

```
mysql> explain select * From sbtest1 where id > 1000;
+-------------------------------------------------------------------------------------------------
----------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------------
----------------------------+
| UnionAll(concurrent=true)
|
|   LogicalView (tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM
`sbtest1` WHERE (`id` > ?)") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------------
----------------------------+
3 rows in set (0.00 sec)
```

LogicalView consists of three parts of information:

- **tables:** the name of the underlying data source table. The value uses a period (`.`) as a separator, which is preceded by the number of the database shard and followed by the name and number of a table shard. Consecutive numbers will be shortened, for example, `[000-127]` , indicating all table shards with numbers ranging from `000` to `127` .
- **shardCount:** the total number of table shards that you want to access. In this example, 128 table shards with numbers ranging from `000` to `127` will be queried.
- **sql:** the SQL template sent to the underlying data source. The value in the example is for reference only. PolarDB-X 1.0 replaces the table name with the physical table name during execution and replaces the constant `10` with a question mark (`?`). This is because PolarDB-X 1.0 enables PlanCache by default and parameterizes SQL statements.

UnionAll

UnionAll corresponds to `UNION ALL` . Generally, this operator has multiple inputs and a UNION operation is performed on the inputs. In the preceding example, UnionAll on LogicalView means that UNION is performed on the data in all table shards.

The `concurrent` in UnionAll indicates whether to run its child operators in parallel. Default value: true.

UnionDistinct

Similar to UnionAll, UnionDistinct corresponds to `UNION DISTINCT` . For example:

```
mysql> explain select * From sbtest1 where id > 1000 union distinct select * From sbtest1 w
here id < 200;
+-------------------------------------------------------------------------------------------
------------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------
------------------------------+
| UnionDistinct(concurrent=true)
|
|   UnionAll(concurrent=true)
|
|     LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FRO
M `sbtest1` WHERE (`id` > ?)") |
|   UnionAll(concurrent=true)
|
|     LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FRO
M `sbtest1` WHERE (`id` < ?)") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------
------------------------------+
6 rows in set (0.02 sec)
```

MergeSort

MergeSort is the merge sort operator. Generally, this operator has multiple child operators. PolarDB-X 1.0 implements merging sorting for ordered data and memory sorting for unordered data. For example:

```
mysql> explain select *from sbtest1 where id > 1000 order by id limit 5,10;
+-------------------------------------------------------------------------------------------
-----------------------------------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------
-----------------------------------------------------+
| MergeSort(sort="id ASC", offset=?1, fetch=?2)
|
|   LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM
`sbtest1` WHERE (`id` > ?) ORDER BY `id` LIMIT (? + ?)") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------
-----------------------------------------------------+
3 rows in set (0.00 sec)
```

The MergeSort operator consists of three parts of information:

- **sort**: the sort field and sort order. Specifically, `id ASC` specifies that data is sorted in ascending order based on the `id` field, and `DESC` specifies that data is sorted in descending order.
- **offset**: the offset to obtain the result set. Similarly, due to the parameterization of SQL statements, the `offst` in the example is expressed as `?1`, where the question mark (`?`) is a

dynamic parameter, and the number that follows corresponds to the subscript of the parameter list. In this example, the parameter corresponding to the SQL statement is `[1000, 5, 10]`, and therefore, the actual value of `?1` is `5`.

- **fetch:** the maximum number of returned data rows. Similar to `offset`, this parameter is also parameterized. The actual value is `10`.

Aggregate

Aggregate is an aggregate operator, which consists of two parts: the Group By field and the aggregate function. For example:

```
mysql> explain select k, count(*) from sbtest1 where id > 1000 group by k;
+-------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| Aggregate(group="k", count(*)="SUM(count(*))")
|
|   MergeSort(sort="k ASC")
|
|     LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT `k`,
COUNT(*) AS `count(*)` FROM `sbtest1` WHERE (`id` > ?) GROUP BY `k` ORDER BY `k`") |
| HitCache:true
|
+-------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

Aggregate consists of two parts of information:

- **group:** the GROUP BY field, which is `k` in this example.
- **Aggregate function:**: The equal sign (`=`) follows the output column name corresponding to the aggregate function and is followed by the corresponding calculation method. In `count(*)="SUM(count(*))"` of the example, the first `count(*)` corresponds to the output column name. The following `SUM(count(*))` means that the final results of the `count(*)` column is obtained by performing a `SUM` operation on the input data of the `count(*)`.

This indicates that PolarDB-X 1.0 divides aggregate operations into two parts. First, the aggregate operations are pushed down to the underlying data sources for local aggregation. Then, the global aggregation of the locally aggregated results is performed at the PolarDB-X 1.0 layer. The final aggregation of PolarDB-X 1.0 is based on sorting. Therefore, a child operator `Sort` is added in the optimizer, and the `Sort` operator is further converted to `MergeSort` by pushdown.

Another example of the `AVG` aggregate function is as follows:

```
mysql> explain select k, avg(id) avg_id from sbtest1 where id > 1000 group by k;
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
----------------------------+
| LOGICAL PLAN|
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
----------------------------+
| Project(k="k", avg_id="sum_pushed_sum / sum_pushed_count")|
|   Aggregate(group="k", sum_pushed_sum="SUM(pushed_sum)", sum_pushed_count="SUM(pushed_cou
nt)")|
|     MergeSort(sort="k ASC")|
|       LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT `k`
, SUM(`id`) AS `pushed_sum`, COUNT(`id`) AS `pushed_count` FROM `sbtest1` WHERE (`id` > ?)
GROUP BY `k` ORDER BY `k`")|
| HitCache:false|
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
----------------------------+
5 rows in set (0.01 sec)
```

PolarDB-X 1.0 converts the `AVG` aggregate function to `SUM` or `COUNT`, and then converts it to local aggregation and global aggregation respectively based on the push rules of `SUM` and `COUNT`. You can try to understand the execution plans of other aggregate functions.

Note PolarDB-X 1.0 converts the DISTINCT operation to the GROUP operation as follows:

```
mysql> explain select distinct k from sbtest1 where id > 1000;
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| Aggregate(group="k")
|
|   MergeSort(sort="k ASC")
|
|     LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT `
k` FROM `sbtest1` WHERE (`id` > ?) GROUP BY `k` ORDER BY `k`") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
4 rows in set (0.02 sec)
```

TmpSort

TmpSort sorts data in memory. The difference from MergeSort is that MergeSort can have multiple child operators, and the data returned by each child operator has been sorted. TmpSort has only one child operator.

The query plan information for TmpSort is consistent with that for MergeSort. For more information, see MergeSort.

Project

The Project operator indicates a projection operation to select some columns from the input data for output or to convert some columns (by using a function or expression computation) for output. The Project operator can also contain constants. In the preceding `AVG` example, the top-level is a `Project`, and its output is `k` and `sum_pushed_sum/sum_pushed_count`. The latter corresponds to a column named `avg_id`.

```
mysql> explain select 'Hello, DRDS', 1 / 2, CURTIME();
+-------------------------------------------------------------------------------------+
| LOGICAL PLAN                                                                        |
+-------------------------------------------------------------------------------------+
Project(Hello, DRDS="_UTF-16'Hello, DRDS'", 1 / 2="1 / 2", CURTIME()="CURTIME()")
|                                                                                     |
| HitCache:false                                                                      |
+-------------------------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

The Project plan includes the name of each column and the corresponding columns, values, functions, and expressions.

Filter

The Filter operator performs a filtering operation that contains some filter conditions. This operator performs filtering on the input data. The data that meets the filter conditions is output and the remaining data is discarded. The following example includes most of the operators described previously and therefore is rather complex.

```
mysql> explain select k, avg(id) avg_id from sbtest1 where id > 1000 group by k having avg_
id > 1300;
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------+
| LOGICAL PLAN  |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------+
| Filter(condition="avg_id > ?1")  |
|   Project(k="k", avg_id="sum_pushed_sum / sum_pushed_count")  |
|     Aggregate(group="k", sum_pushed_sum="SUM(pushed_sum)", sum_pushed_count="SUM(pushed_c
ount)")  |
|       MergeSort(sort="k ASC")  |
|         LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT `
k`, SUM(`id`) AS `pushed_sum`, COUNT(`id`) AS `pushed_count` FROM `sbtest1` WHERE (`id` > ?
) GROUP BY `k` ORDER BY `k`") |
| HitCache:false  |
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------+
6 rows in set (0.01 sec)
```

Based on the SQL of the preceding `AVG` example, `having avg_id > 1300` is added. A Filter operator is added at the top of the execution plan to filter all data that satisfies `avg_id > 1300`.

You may ask why the condition in WHERE has no corresponding Filter operator? At a stage of the PolarDB-X 1.0 optimizer, the Filter operator of the WHERE condition does exist, but it is finally pushed down to LogiacalView. Therefore, you can find `id > 1000` in LogicalView's SQL.

NlJoin

NlJoin is the NestLoop Join operator, which allows you to use the NestLoop method to join two tables. PolarDB-X 1.0 implements two JOIN policies: NlJoin and BKAJoin. The latter refers to Batched Key Access Join. When you query data by using key-value pairs, a batch of data is retrieved from the left table. An IN condition is concatenated into the SQL statement for accessing the right table to obtain a batch of data from the right table at a time.

```
mysql> explain select a.* from sbtest1 a, sbtest1 b where a.id = b.k and a.id > 1000;
+-----------------------------------------------------------------------------------------------
---------------------------------+
| LOGICAL PLAN
|
+-----------------------------------------------------------------------------------------------
---------------------------------+
| Project(id="id", k="k", c="c", pad="pad")
|
|   NlJoin(id="id", k="k", c="c", pad="pad", k0="k0", condition="id = k", type="inner")
|
|     UnionAll(concurrent=true)
|
|       LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * F
ROM `sbtest1` WHERE (`id` > ?)")  |
|     UnionAll(concurrent=true)
|
|       LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT `k`
FROM `sbtest1` WHERE (`k` > ?)") |
| HitCache:false
|
+-----------------------------------------------------------------------------------------------
---------------------------------+
7 rows in set (0.03 sec)
```

The NlJOIN plan includes three parts:

- **Output column info:** the output column name. In this example, the JOIN statement returns five columns. `id="id", k="k", c="c", pad="pad", k0="k0"`.
- **condition:** the join condition. In this example, the join condition is `id = k`.
- **type:** the connection type. In this example, the type is INNER JOIN. Therefore, the connection type is `inner`.

BKAJoin

BKAJoin: JOIN is performed by using batch key-value queries. That is, a batch of data is retrieved from the left table. An IN condition is concatenated into the SQL statement for accessing the right table to obtain a batch of data from the right table at a time.

```
mysql> explain select a.* from sbtest1 a, sbtest1 b where a.id = b.k order by a.id;
+-----------------------------------------------------------------------------------------------
-------------------------------------+
| LOGICAL PLAN
|
+-----------------------------------------------------------------------------------------------
-------------------------------------+
| Project(id="id", k="k", c="c", pad="pad")
|
|   BKAJoin(id="id", k="k", c="c", pad="pad", id0="id0", k0="k0", c0="c0", pad0="pad0", con
dition="id = k", type="inner")       |
|     MergeSort(sort="id ASC")
|
|       LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * F
ROM `sbtest1` ORDER BY `id`")        |
|     UnionAll(concurrent=true)
|
|       LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * F
ROM `sbtest1` WHERE (`k` IN ('?'))") |
| HitCache:false
|
+-----------------------------------------------------------------------------------------------
-------------------------------------+
7 rows in set (0.01 sec)
```

The plan content of BKAJoin is the same as that of NlJoin. The two operators have different names and are designed to inform the executor of the method used to perform the JOIN operation. In addition, `'k' IN ('?')` in LogicalView on the right table in the preceding execution plan is an IN query template created by the optimizer for querying data in the right table.

LogicalModifyView

As mentioned above, the LogicalView operator obtains data from the underlying data source. Correspondingly, the LogicalModifyView operator modifies the underlying data source and also includes an SQL statement. This SQL statement may be an INSERT, UPDATE, or DELETE statement.

```
mysql> explain update sbtest1 set c='Hello, DRDS' where id > 1000;
+-------------------------------------------------------------------------------------------------
-------------------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------------
-------------------------------------+
| LogicalModifyView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="UPDATE `sb
test1` SET `c` = ? WHERE (`id` > ?)") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------------
-------------------------------------+
2 rows in set (0.03 sec)
mysql> explain delete from sbtest1 where id > 1000;
+-------------------------------------------------------------------------------------------------
-------------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------------
-------------------------------+
| LogicalModifyView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="DELETE FRO
M `sbtest1` WHERE (`id` > ?)") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------------
-------------------------------+
2 rows in set (0.03 sec)
```

The query plan of the LogicalModifyView operator is similar to that of the LogicalView operator, including the delivered physical table shards, the number of table shards, and an SQL template. Similarly, PlanCache is enabled, thus the SQL statement is parameterized and constants in the SQL template are replaced with question marks (`?`).

PhyTableOperation

PhyTableOperation: performs an operation on a physical table shard. This operator is used only in INSERT INTO... VALUES ....

```
mysql> explain insert into sbtest1 values(1, 1, '1', '1'),(2, 2, '2', '2');
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-------------------------+
| LOGICAL PLAN
|
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-------------------------+
| PhyTableOperation(tables="SYSBENCH_CORONADB_1526954857179TGMMSYSBENCH_CORONADB_VGOC_0000_
RDS.[sbtest1_001]", sql="INSERT INTO ? (`id`, `k`, `c`, `pad`) VALUES(?, ?, ?, ?)", params=
"`sbtest1_001`,1,1,1,1") |
| PhyTableOperation(tables="SYSBENCH_CORONADB_1526954857179TGMMSYSBENCH_CORONADB_VGOC_0000_
RDS.[sbtest1_002]", sql="INSERT INTO ? (`id`, `k`, `c`, `pad`) VALUES(?, ?, ?, ?)", params=
"`sbtest1_002`,2,2,2,2") |
|
|
| HitCache:false
|
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-------------------------+
4 rows in set (0.00 sec)
```

In this example, the INSERT statement is executed to insert two rows of data, with each row of data corresponding to one PhyTableOperation operator. The PhyTableOperation operator consists of the following parts of information:

- **tables:** the name of a physical table. Only one physical table name is specified.
- **sql:** the SQL template. Table names and constants in the SQL template are all parameterized and replaced with question marks (`?`) and the corresponding parameters are listed in the params parameter.
- **params:** the actual parameters corresponding to the question marks (?) in the SQL template, including table names and constants.

Other information

HitCache

PolarDB-X 1.0 enables PlanCache by default. HitCache is used to inform you about whether the query hits PlanCache. In the following example, HitCache is set to false for the first run and true for the second run.

```
mysql> explain select * From sbtest1 where id > 1000;
+-------------------------------------------------------------------------------------------------
-----------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------------
-----------------------------+
| UnionAll(concurrent=true)
|
|   LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM
`sbtest1` WHERE (`id` > ?)") |
| HitCache:false
|
+-------------------------------------------------------------------------------------------------
-----------------------------+
3 rows in set (0.01 sec)
mysql> explain select * From sbtest1 where id > 1000;
+-------------------------------------------------------------------------------------------------
-----------------------------+
| LOGICAL PLAN
|
+-------------------------------------------------------------------------------------------------
-----------------------------+
| UnionAll(concurrent=true)
|
|   LogicalView(tables="[0000-0031].sbtest1_[000-127]", shardCount=128, sql="SELECT * FROM
`sbtest1` WHERE (`id` > ?)") |
| HitCache:true
|
+-------------------------------------------------------------------------------------------------
-----------------------------+
3 rows in set (0.00 sec)
```

16.Error codes

This topic describes the common error codes that may be returned in PolarDB-X 1.0 and how to troubleshoot the errors.

TDDL-4006 ERR_TABLE_NOT_EXIST

The error code is returned because the specified data table does not exist.

Example:

```
ERR-CODE: [TDDL-4006][ERR_TABLE_NOT_EXIST] Table '*****' doesn't exist.
```

This error code indicates that the data table does not exist in PolarDB-X 1.0 or PolarDB-X 1.0 has failed to load the metadata of the data table due to unknown reasons.

If this error is returned, Submit a ticket.

TDDL-4007 ERR_CANNOT_FETCH_TABLE_META

The error code is returned because PolarDB-X 1.0 has failed to load the metadata of a data table.

Example:

```
ERR-CODE: [TDDL-4007][ERR_CANNOT_FETCH_TABLE_META] Table '*****' metadata
cannot be fetched because Table '*****.*****' doesn't exist.
```

This error code indicates that PolarDB-X 1.0 has failed to query the metadata of the data table. This error may occur due to one of the following reasons:

- The data table is not created.
- The data table in the database shard is manually deleted or renamed.
- PolarDB-X 1.0 cannot connect to the backend ApsaraDB RDS for MySQL instances.

If this error is returned, check whether the specified data table exists and confirm whether the status of all backend ApsaraDB RDS for MySQL instances of PolarDB-X 1.0 is normal.

If the data table is manually deleted or renamed, you can use the data restoration feature of ApsaraDB RDS for MySQL to restore the data. If the error persists, Submit a ticket.

TDDL-4100 ERR_ATOM_NOT_AVALILABLE

The error code is returned because a backend ApsaraDB RDS for MySQL instance of PolarDB-X 1.0 is unavailable.

Example:

```
ERR-CODE: [TDDL-4100][ERR_ATOM_NOT_AVALILABLE] Atom : ***** isNotAvailable
```

If PolarDB-X 1.0 detects that the status of an ApsaraDB RDS for MySQL instance on the backend is abnormal, PolarDB-X 1.0 temporarily blocks access to the instance and returns the TDDL-4100 error. If this error is returned, check whether the status of all backend ApsaraDB RDS for MySQL instances of PolarDB-X 1.0 is abnormal. If an abnormal ApsaraDB RDS for MySQL instance is detected, recover the related instance.

After the ApsaraDB RDS for MySQL instance is recovered, PolarDB-X 1.0 automatically changes the state of the instance and allows applications to access the instance.

TDDL-4101 ERR_ATOM_GET_CONNECTION_FAILED_UNKNOWN_REASON

The error code is returned because PolarDB-X 1.0 has failed to connect to a backend ApsaraDB RDS for MySQL instance due to unknown reasons.

Example:

```
ERR-CODE: [TDDL-4101][ERR_ATOM_GET_CONNECTION_FAILED_UNKNOWN_REASON] Get
connection for db '*****' from pool failed. AppName:*****, Env:*****,
UnitName:null. Message from pool: wait millis 5000, active 0, maxActive 5.
You should look for the following logs which contains the real reason.
```

When PolarDB-X 1.0 processes requests, PolarDB-X 1.0 asynchronously establishes connections to the backend ApsaraDB RDS for MySQL instances. If PolarDB-X fails to connect to a backend ApsaraDB RDS for MySQL instance within a period of time and no error causes are returned for the asynchronous task, PolarDB-X 1.0 returns the TDDL-4101 error to the application.

In most cases, this error is returned because the status of the backend ApsaraDB RDS for MySQL instance is abnormal. If this error persists after the backend ApsaraDB RDS for MySQL instance is recovered, Submit a ticket.

TDDL-4102 ERR_ATOM_GET_CONNECTION_FAILED_KNOWN_REASON

The error code is returned because PolarDB-X 1.0 has failed to connect to a backend ApsaraDB RDS for MySQL instance due to known reasons.

Example:

```
ERR-CODE: [TDDL-4102][ERR_ATOM_GET_CONNECTION_FAILED_KNOWN_REASON] Get
connection for db '*****' failed because wait millis 5000, active 0,
maxActive 5
```

This error code is returned if an error occurs when PolarDB-X 1.0 connects to a backend ApsaraDB RDS for MySQL instance. The error causes are included in the `ERR-CODE` message.

PolarDB-X 1.0 may fail to connect to a backend ApsaraDB RDS for MySQL instance due to one of the following reasons:

- The number of connections to the backend ApsaraDB RDS for MySQL instance has reached the upper limit.
- The connection to the backend ApsaraDB RDS for MySQL instance has timed out.
- The connection to the backend ApsaraDB RDS for MySQL instance is rejected.

If this error persists after you troubleshoot the issues on the backend ApsaraDB RDS for MySQL instance, Submit a ticket.

TDDL-4103 ERR_ATOM_CONNECTION_POOL_FULL

The error code is returned because the connection pool of the backend ApsaraDB RDS for MySQL instances of PolarDB-X 1.0 is full.

Example:

```
ERR-CODE: [TDDL-4103][ERR_ATOM_CONNECTION_POOL_FULL] Pool of DB '*****' is
full. Message from pool: wait millis 5000, active 5, maxActive 5.
AppName:*****, Env:*****, UnitName:null.
```

This error code indicates that the backend connection pool of PolarDB-X 1.0 is full. The TDDL-4103 error may be returned due to one of the following reasons:

- The execution of SQL statements that are sent from an application is slow, and the operation is performed over a connection for a long period of time. As a result, the number of available connections is insufficient.
- An application does not close the connections to a database. This causes connection leaks.
- A large number of cross-database queries are performed in parallel. This operation is performed over a large number of connections. The cross-database queries include the queries for aggregation and statistical analysis and the queries for data in databases that are not sharded.

To resolve this error, we recommend that you use the following methods:

- Use frameworks such as Spring JDBC and MyBatis to connect to databases.
- Optimize SQL queries based on the performance analysis reports and suggestions of database administrators.
- Use the read/write splitting feature of PolarDB-X 1.0 to forward cross-database queries to read-only nodes.
- Upgrade the specifications of your ApsaraDB RDS for MySQL instances to improve the backend processing performance.
- Submit a ticket to change the maximum number of backend connections for your PolarDB-X 1.0 instance.

TDDL-4104 ERR_ATOM_CREATE_CONNECTION_TOO_SLOW

The error code is returned because the connection to a backend ApsaraDB RDS for MySQL instance of a PolarDB-X 1.0 instance has timed out.

Example:

```
ERR-CODE: [TDDL-4104][ERR_ATOM_CREATE_CONNECTION_TOO_SLOW] Get connection
for db '*****' from pool timeout. AppName:*****, Env:*****, UnitName:null.
Message from pool: wait millis 5000, active 3, maxActive 5.
```

When PolarDB-X 1.0 connects to a backend ApsaraDB RDS for MySQL instance in an asynchronously manner, the connection times out if a large number of connection requests are sent in a short period of time or it takes a long time to establish a connection to the backend ApsaraDB RDS for MySQL instance.

In most cases, this error occurs due to the heavy workloads on the backend ApsaraDB RDS for MySQL instance. To resolve this error, we recommend that you use the read/write splitting feature of PolarDB-X 1.0 or upgrade the specifications of the ApsaraDB RDS for MySQL instance.

If this error persists after you troubleshoot the issues on the backend ApsaraDB RDS for MySQL instance, Submit a ticket.

If this error occurs because a large number of connection requests are sent in a short period of time, Submit a ticket to change the minimum number of connections for your PolarDB-X 1.0 instance.

TDDL-4105 ERR_ATOM_ACCESS_DENIED

The error code is returned because the connection request that PolarDB-X 1.0 sent to a backend ApsaraDB RDS for MySQL instance is rejected.

Example:

```
ERR-CODE: [TDDL-4105][ERR_ATOM_ACCESS_DENIED] DB '*****' Access denied for
user '*****'@'*****'. AppName:*****, Env:*****, UnitName:null. Please
contact DBA to check.
```

This error code indicates that the access request that includes a username and a password from PolarDB-X 1.0 to the ApsaraDB RDS for MySQL instance is rejected.

If the username or password that is automatically created by PolarDB-X 1.0 is changed on the backend ApsaraDB RDS for MySQL instance, PolarDB-X 1.0 returns the TDDL-4105 error for the access request. To resolve the error, Submit a ticket to rectify the username or the password of your PolarDB-X 1.0 instance.

PolarDB-X 1.0 also returns the TDDL-4105 error if the backend ApsaraDB RDS for MySQL instance expires or if an overdue payment occurs in your account. In this case, renew the instance at your earliest opportunity.

TDDL-4106 ERR_ATOM_DB_DOWN

The error code is returned because PolarDB-X 1.0 has failed to connect to a backend ApsaraDB RDS for MySQL instance.

Example:

```
ERR-CODE: [TDDL-4106][ERR_ATOM_DB_DOWN] DB '*****' cannot be connected.
AppName:*****, Env:*****, UnitName:null. It seems a very real possibility
that this DB IS DOWN. Please contact DBA to check.
```

This error code indicates that the connection request from PolarDB-X 1.0 to the backend ApsaraDB RDS for MySQL instance has timed out or no response is returned for the connection request. In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instance. To resolve the error, Submit a ticket.

TDDL-4108 ERR_VARIABLE_CAN_NOT_SET_TO_NULL_FOR_NOW

The error code is returned because the value of a variable cannot be set to `NULL`.

Example:

```
ERR-CODE: [TDDL-4108][ERR_VARIABLE_CAN_NOT_SET_TO_NULL_FOR_NOW] System
variable ***** can''t set to null for now;
```

You cannot execute the `SET` statement to set the value of some MySQL variables to `NULL`. If the value of such a variable is set to NULL, PolarDB-X 1.0 returns the TDDL-4108 error.

If this error occurs, check the value of the variable and rectify the value based on the official documentation of MySQL. For more information, see Server System Variables.

TDDL-4200 ERR_GROUP_NOT_AVALILABLE

The error code is returned because a PolarDB-X 1.0 database shard is unavailable.

Example:

```
ERR-CODE: [TDDL-4200][ERR_GROUP_NOT_AVALILABLE] The TDDL Group ***** is
running in fail-fast status, caused by this SQL:***** which threw a fatal
exception as *****.
```

If the backend ApsaraDB RDS for MySQL instance in which the database shard resides cannot be accessed and no other instances are available for the database shard, PolarDB-X 1.0 sets the status of the database shard to `fail-fast` and returns the TDDL-4200 error.

In most cases, this error occurs because the backend ApsaraDB RDS for MySQL instance fails. Troubleshoot the failure based on the error information. After the ApsaraDB RDS for MySQL instance is recovered, PolarDB-X 1.0 automatically changes the state of the instance from `fail-fast`.

If this error persists after you resolve the fault in the backend ApsaraDB RDS for MySQL instance, Submit a ticket.

TDDL-4201 ERR_GROUP_NO_ATOM_AVALILABLE

The error code is returned because no ApsaraDB RDS for MySQL instances are available for a PolarDB-X 1.0 database shard.

Example:

```
ERR-CODE: [TDDL-4201][ERR_GROUP_NO_ATOM_AVALILABLE] All weights of DBs in
Group '*****' is 0. Weights is: *****.
```

When all ApsaraDB RDS for MySQL instances in which a database shard resides are unavailable or the database shard is in the `fail-fast` state, PolarDB-X 1.0 returns the TDDL-4201 error.

In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instances. Check the status of all backend ApsaraDB RDS for MySQL instances and resolve the fault. If the error persists, Submit a ticket.

TDDL-4202 ERR_SQL_QUERY_TIMEOUT

The error code is returned because a query in PolarDB-X 1.0 has timed out.

Example:

```
ERR-CODE: [TDDL-4202][ERR_SQL_QUERY_TIMEOUT] Slow query leads to a timeout
exception, please contact DBA to check slow sql. SocketTimout:*** ms,
Atom:*****, Group:*****, AppName:*****, Env:*****, UnitName:null.
```

This error code indicates that the execution duration of the SQL statement on the backend ApsaraDB RDS for MySQL instances exceeds the value of the `socketTimeout` parameter that you specified for your PolarDB-X 1.0 instance. The default value of the socketTimeout parameter is 900 seconds for your PolarDB-X 1.0 instance.

We recommend that you optimize the SQL statement and create suitable indexes on the backend ApsaraDB RDS for MySQL instances to improve the SQL query performance.

If the error persists after the SQL statement is optimized, use the following PolarDB-X 1.0 hint syntax to specify a temporary timeout period:

```
/*TDDL:SOCKET_TIMEOUT=900000*/ SELECT * FROM dual;
```

Specify the value of the `SOCKET_TIMEOUT` parameter in milliseconds.

For more information about PolarDB-X 1.0 hints, see Specify a custom time-out period for an SQL statement.

To permanently change the timeout period for PolarDB-X 1.0, Submit a ticket.

TDDL-4203 ERR_SQL_QUERY_MERGE_TIMEOUT

The error code is returned because a distributed query has timed out.

Example:

```
ERR-CODE: [TDDL-4203][ERR_SQL_QUERY_MERGE_TIMEOUT] Slow sql query leads to
a timeout exception during merging results, please optimize the slow sql.
The the default timeout is *** ms. DB is *****
```

When you query distributed data in PolarDB-X 1.0, the default timeout period is 900 seconds.

This error code indicates that the system has scanned the data in multiple database shards to execute the SQL statement and the execution duration is longer than 900 seconds. To optimize the SQL statement, perform the following steps:

- Include a shard key in the WHERE clause to specify a database shard on which you want to execute the SQL statement.
- Check whether a suitable index can be created on the backend ApsaraDB RDS for MySQL instances. Indexes can improve the query performance of a database shard.
- Eliminate time-consuming operations in the distributed query, such as cross-database JOIN queries and queries that are performed based on data resorting. This helps reduce the number of resources that are consumed during data merge operations.

If the error persists after the SQL statement is optimized, use the following hint syntax to specify a temporary timeout period for PolarDB-X 1.0:

```
/*TDDL:SOCKET_TIMEOUT=900000*/ SELECT * FROM dual;
```

Specify the value of the `SOCKET_TIMEOUT` parameter in milliseconds.

For more information about PolarDB-X 1.0 hints, see Specify a custom time-out period for an SQL statement.

If this error persists, Submit a ticket.

TDDL-4400 ERR_SEQUENCE

The error code is returned because a sequence has failed to be processed.

Example:

```
ERR-CODE: [TDDL-4400][ERR_SEQUENCE] Sequence : All dataSource faild to get
value!
```

This error code indicates that PolarDB-X 1.0 has failed to process the sequence. The error message is provided after `Sequence :` .

In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instances. As a result, data tables that are related to the sequence cannot be accessed. We recommend that you check the status of all backend ApsaraDB RDS for MySQL instances. If this error persists after you resolve the fault in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-4401 ERR_MISS_SEQUENCE

The error code is returned because the specified sequence does not exist.

Example:

```
ERR-CODE: [TDDL-4401][ERR_MISS_SEQUENCE] Sequence '*****' is not found
```

This error code indicates that the sequence that you specified in the statement does not exist. We recommend that you execute the `SHOW SEQUENCES` statement to query the name of each sequence that you created in PolarDB-X 1.0 and specify a valid sequence name.

If the sequence that you want to use does not exist, you can use the following `CREATE SEQUENCE` syntax to create the sequence:

```
CREATE SEQUENCE <sequence name> [ START WITH <numeric value> ]
[ INCREMENT BY <numeric value> ] [ MAXVALUE <numeric value> ]
[ CYCLE | NOCYCLE ]`
```

If the sequence that you specified exists and the TDDL-4401 error persists, Submit a ticket.

For more information about sequences, see Sequence.

TDDL-4403 ERR_MISS_SEQUENCE_TABLE_ON_DEFAULT_DB

The error code is returned because the data table that corresponds to a sequence does not exist.

Example:

```
ERR-CODE: [TDDL-4403][ERR_MISS_SEQUENCE_TABLE_ON_DEFAULT_DB] Sequence table
is not in default db.
```

This error code indicates that the data table named `sequence` or `sequence_opt` cannot be found in the backend database. To troubleshoot the error, Submit a ticket.

TDDL-4404 ERR_SEQUENCE_TABLE_META

The error code is returned because the schema of the data table that corresponds to a sequence is invalid.

Example:

```
ERR-CODE: [TDDL-4404][ERR_SEQUENCE_TABLE_META] the meta of sequence table
is error, some columns missed
```

This error code indicates that specific fields are missing in the data table that corresponds to the sequence. This data table can be the `sequence` or `sequence_opt` table. To troubleshoot the error, Submit a ticket.

TDDL-4405 ERR_INIT_SEQUENCE_FROM_DB

The error code is returned because a sequence has failed to be initialized.

Example:

```
ERR-CODE: [TDDL-4405][ERR_INIT_SEQUENCE_FROM_DB] init sequence manager
error: *****
```

This error code indicates that the system has failed to initialize the sequence that you want to use. The error message is provided after `init sequence manager error:` .

We recommend that you check the status of all backend ApsaraDB RDS for MySQL instances. If this error persists after you resolve the faults in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-4407 ERR_OTHER_WHEN_BUILD_SEQUENCE

The error code is returned because the data table that corresponds to a sequence cannot be accessed.

Example:

```
ERR-CODE: [TDDL-4407][ERR_OTHER_WHEN_BUILD_SEQUENCE] error when build
sequence: *****
```

This error code is returned if an error occurs when you access a data table that corresponds to the sequence, such as the `sequence` or `sequence_opt` table. The error message is provided after `error when build sequence:` .

We recommend that you check the status of all backend ApsaraDB RDS for MySQL instances. If this error persists after you resolve the faults in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

DDL-4408 ERR_SEQUENCE_NEXT_VALUE

The error code is returned because the system has failed to obtain the values in a sequence.

Example:

```
ERR-CODE: [TDDL-4408][ERR_SEQUENCE_NEXT_VALUE] error when get sequence's
next value, sequence is: *****, error: *****
```

This error code is returned if an error occurs when you obtain the values of a sequence by using a PolarDB-X 1.0 auto-increment primary key or the `<sequence name>.NEXTVAL` syntax. The cause of error is provided after `error:` .

In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instances. We recommend that you check the status of and access workloads on the backend ApsaraDB RDS for MySQL instances. If this error persists after you resolve the fault in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-4500 ERR_PARSER

The error code is returned because the SQL statement has failed to be parsed.

Example:

```
ERR-CODE: [TDDL-4500][ERR_PARSER] not support statement: '*****'
```

PolarDB-X 1.0 supports the SQL syntax that complies with the SQL-92 standard and the extended syntax and functions that are supported by MySQL. Check whether the SQL statement that you executed complies with the standard SQL syntax and MySQL specifications that are supported by PolarDB-X 1.0.

For more information about the standard SQL syntax, see Standard SQL syntax.

For more information about SQL statements that are compatible with PolarDB-X 1.0, see SQL limits.

For more information about the SQL syntax in MySQL 5.6, see SQL syntax in MySQL 5.6.

If this error persists after you rectify the SQL statement, Submit a ticket.

TDDL-4501 ERR_OPTIMIZER

The error code is returned because the optimizer has failed to convert an SQL statement.

Example:

```
ERR-CODE: [TDDL-4501][ERR_OPTIMIZER] optimize error by: Unknown column
'*****' in 'order clause'
```

The optimizer of PolarDB-X 1.0 can convert an SQL statement to an internal syntax tree. If a logic error occurs in an SQL statement, the optimizer fails to convert the SQL statement. In this case, the TDDL-4501 error is returned.

We recommend that you check and modify the SQL statement based on the cause of error. The cause of error is provided after `optimize error by:`. If this error persists after you modify the SQL statement, Submit a ticket.

TDDL-4502 ERR_OPTIMIZER_MISS_ORDER_FUNCTION_IN_SELECT

The error code is returned because the `SELECT` clause does not contain the columns that are returned by the function specified in the `ORDER BY` clause.

Example:

```
ERR-CODE: [TDDL-4502][ERR_OPTIMIZER_MISS_ORDER_FUNCTION_IN_SELECT] Syntax
Error: orderBy/GroupBy Column ***** is not existed in select clause`
```

In PolarDB-X 1.0, if the `ORDER BY` clause contains a function that returns columns, such as RAND(), the returned columns must also be specified in the `SELECT` clause. If the SELECT clause does not contain the returned columns, the TDDL-4502 error is returned.

We recommend that you include the corresponding columns in the `SELECT` clause.

TDDL-4504 ERR_OPTIMIZER_SELF_CROSS_JOIN

The error code is returned because an SQL statement does not meet the conditions that are required to perform a SELF JOIN query on a table.

Example:

```
ERR-CODE: [TDDL-4504][ERR_OPTIMIZER_SELF_CROSS_JOIN] self cross join case,
add shard column filter on right table
```

When PolarDB-X 1.0 performs a SELF JOIN query on a table, the TDDL-4504 error is returned if the `WHERE` clause includes only the shard key of the left table or the right table.

We recommend that you include the shard keys of the left table and the right table in the `WHERE` clause in the SQL statement.

TDDL-4506 ERR_MODIFY_SHARD_COLUMN

The error code is returned because shard keys cannot be updated.

Example:

```
ERR-CODE: [TDDL-4506][ERR_MODIFY_SHARD_COLUMN] Column '*****' is a sharding
key of table '*****', which is forbidden to be modified.
```

PolarDB-X 1.0 forbids you to change the value of a shard key by using the UPDATE statement. Update operations may change the shard where data resides. Therefore, PolarDB-X 1.0 cannot ensure data consistency and the atomicity of operations.

We recommend that you execute the `DELETE and INSERT` statements that have the same effect as the `UPDATE` statement to change the value of a shard key.

TDDL-4508 ERR_OPTIMIZER_NOT_ALLOWED_SORT_MERGE_JOIN

The error code is returned because the sort merge join operation cannot be performed.

Example:

```
ERR-CODE: [TDDL-4508][ERR_OPTIMIZER_NOT_ALLOWED_SORT_MERGE_JOIN] sort merge
join is not allowed when missing equivalent filter
```

If the data tables on which you want to perform a join operation by executing an SQL statement are stored in different ApsaraDB RDS for MySQL instances, PolarDB-X 1.0 uses the `sort-merge join` algorithm. This algorithm can be used only if you specify the same join conditions for the left table and the right table in the SQL statement. If the join conditions that you specify for the left table are different from the join conditions that you specify for the right table, the TDDL-4508 error is returned.

We recommend that you include the equivalent JOIN conditions in the `JOIN` or `WHERE` clause in the SQL statement.

TDDL-4509 ERR_OPTIMIZER_ERROR_HINT

The error code is returned because the hint syntax is invalid.

Example:

```
ERR-CODE: [TDDL-4509][ERR_OPTIMIZER_ERROR_HINT] Hint Syntax Error:
unexpected operation: *****.
```

This error code indicates that the syntax of the hint that you include in the SQL statement cannot be parsed by PolarDB-X 1.0. For more information about the hint syntax, see Overview.

TDDL-4510 ERR_CONTAINS_NO_SHARDING_KEY

The error code is returned because a shard key is not specified in an SQL statement.

Example:

```
ERR-CODE: [TDDL-4510][ERR_CONTAINS_NO_SHARDING_KEY] Your SQL contains NO
SHARDING KEY '*****' for table '*****', which is not allowed in DEFAULT.
```

If the full table scan feature is not enabled for a PolarDB-X 1.0 table shard, you must include the shard key in the `WHERE` clause to access the table. If the WHERE clause does not contain the shard key, the TDDL-4510 error is returned.

When PolarDB-X 1.0 creates a table, the full table scan feature is enabled by default. If the full table scan feature is manually disabled, make sure that the shard key of the table is specified in each SQL statement that scans the data in the table.

TDDL-4511 ERR_INSERT_CONTAINS_NO_SHARDING_KEY

The error code is returned because a shard key is not specified in the `INSERT` statement.

Example:

```
ERR-CODE: [TDDL-4511][ERR_INSERT_CONTAINS_NO_SHARDING_KEY] Your INSERT SQL
contains NO SHARDING KEY '*****' for table '*****'.
```

In PolarDB-X 1.0, if you want to execute the INSERT statement to insert the data of a sharded table, you must specify the shard key of the table in the INSERT statement unless the shard key is an auto-increment primary key. If the INSERT statement does not contain the shard key, the TDDL-4511 error is returned.

If this error occurs, we recommend that you include the shard key in the `INSERT` statement.

TDDL-4515 ERR_CONNECTION_CHARSET_NOT_MATCH

The error code is returned because the specified character set is not supported.

Example:

```
ERR-CODE: [TDDL-4515][ERR_CONNECTION_CHARSET_NOT_MATCH] Caused by MySQL's
character_set_connection doesn't match your input charset. Partition DDL can
only take ASCII or chinese column name. If you want use chinese table or
column name, Make sure MySQL connection's charset support chinese character.
Use "set names xxx" to set correct charset.
```

PolarDB-X 1.0 supports Chinese characters for table names and field names. The `character_set_connection` parameter specifies the character set that is used by a PolarDB-X 1.0 database to connect to a client. When you execute an SQL statement that contains Chinese characters, the TDDL-4515 error is returned if the character_set_connection parameter is set to a character set that does not support Chinese characters, such as `latin1`.

You can execute the `SHOW VARIABLES LIKE 'character_set_connection'` statement to query the character set that is used by a PolarDB-X 1.0 database to connect to a MySQL client. You can execute the `SET NAMES` statement to change the character set. If you use Java Database Connectivity (JDBC) to connect to a PolarDB-X 1.0 database, configure the `characterEncoding` parameter.

TDDL-4516 ERR_TRUNCATED_DOUBLE_VALUE_OVERFLOW

The error code is returned because an overflow has occurred when the system converts a floating-point number to an integer.

Example:

```
ERR-CODE: [TDDL-4516][ERR_TRUNCATED_DOUBLE_VALUE_OVERFLOW] Truncated
incorrect DOUBLE value '*****' over column[*****]'s value range.
```

This error code indicates that the result is out of the valid range of integers when PolarDB-X 1.0 converts the floating-point number to an integer. We recommend that you check the data types of the specified columns and the input parameters in the SQL statement.

TDDL-4517 ERR_MODIFY_SYSTEM_TABLE

The error code is returned because system tables cannot be modified.

Example:

```
ERR-CODE: [TDDL-4517][ERR_MODIFY_SYSTEM_TABLE] Table '*****' is PolarDB-XSYSTEM
TABLE, which is forbidden to be modified.
```

PolarDB-X 1.0 provides built-in system tables. If you execute an SQL statement to modify the data of a system table, the TDDL-4517 error is returned.

The following system tables cannot be modified: `sequence`, `sequence_opt`, `txc_undo_log`, and `__DRDS__SYSTEM__LOCK__`. Make sure that the names of system tables are not used when you create tables that store business data or when you design a database.

TDDL-4518 ERR_VALIDATE

The error code is returned because the metadata verification has failed.

Example:

```
ERR-CODE: [TDDL-4518][ERR_VALIDATE] Object 'optest1' not found
```

When a PolarDB-X 1.0 compute node receives an SQL statement, the compute node verifies the SQL statement based on existing metadata. This error code indicates that the table or column information that you want to query does not meet the requirements of metadata.

TDDL-4600 ERR_FUNCTION

The error code is returned because an error has occurred for a function call.

Example:

```
ERR-CODE: [TDDL-4600][ERR_FUNCTION] function compute error by Incorrect
parameter count in the call to native function '*****'
```

This error code indicates that the SQL statement uses invalid syntax or contains invalid parameters to call the function. We recommend that you check whether the number and data type of parameters that you use to call the function in the SQL statement are valid.

TDDL-4600 ERR_FUNCTION

The error code is returned because an error has occurred for a function call.

Example:

```
ERR-CODE: [TDDL-4600][ERR_FUNCTION] function compute error by Incorrect
parameter count in the call to native function '*****'
```

This error code indicates that the SQL statement uses invalid syntax or contains invalid parameters to call the function. We recommend that you check whether the number and data type of parameters that you use to call the function in the SQL statement are valid.

TDDL-4601 ERR_EXECUTOR

The error code is returned because an error has occurred when the system executes the SQL statement.

Example:

```
ERR-CODE: [TDDL-4601][ERR_EXECUTOR] only one column is supported in
distinct aggregate
```

This error code is returned if an unexpected error occurs when PolarDB-X 1.0 executes an SQL statement. In most cases, the error occurs because the status of a backend ApsaraDB RDS for MySQL instance is abnormal. We recommend that you check the status of all backend ApsaraDB RDS for MySQL instances. If the error persists after you resolve the faults in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-4602 ERR_CONVERTOR

The error code is returned because the system has failed to convert a data type.

Example:

```
ERR-CODE: [TDDL-4602][ERR_CONVERTOR] convertor error by Unsupported convert:
[*****]
```

This error code indicates that the data type cannot be converted when PolarDB-X 1.0 executes the SQL statement. Check whether the data that is used in the SQL statement requires implicit data type conversion. We recommend that you specify data of the same type for comparison and computing.

TDDL-4603 ERR_ACCROSS_DB_TRANSACTION

The error code is returned because a cross-database transaction has failed.

Example:

```
ERR-CODE: [TDDL-4603][ERR_ACCROSS_DB_TRANSACTION] Transaction accross db is
not supported in current transaction policy, transaction node is: {0}, but
this sql execute on: *****.
```

PolarDB-X 1.0 supports only single-database transactions. All SQL statements for single-database transactions must be forwarded to the same ApsaraDB RDS for MySQL database shard for execution based on the specified forwarding rules. Otherwise, the TDDL-4603 error is returned.

TDDL-4604 ERR_CONCURRENT_TRANSACTION

The error code is returned because a nested transaction has failed.

Example:

```
ERR-CODE: [TDDL-4604][ERR_CONCURRENT_TRANSACTION] Concurrent query is not
supported on transaction group, transaction group is: {0}.
```

PolarDB-X 1.0 does not support nested transactions. If you attempt to start more than two transactions at the same time over the same database connection, the TDDL-4604 error is returned.

We recommend that you do not use nested transactions when you develop applications. You can abstract transactions into a transaction framework at the application layer. This way, no nested transactions are generated.

TDDL-4606 ERR_QUERY_C

The error code is returned because the execution of an SQL statement is canceled.

Example:

```
ERR-CODE: [TDDL-4606][ERR_QUERY_CANCLED] Getting connection is not allowed
when query has been cancled, group is *****
```

When the `KILL` statement is executed to cancel the execution of an SQL statement, PolarDB-X 1.0 returns the TDDL-4606 error for the SQL statement. If this error frequently occurs, check whether the `KILL` statement is executed on a client or a program.

TDDL-4607 ERR_INSERT_WHEN_UPDATE

The error code is returned because an error has occurred when PolarDB-X 1.0 executes the `UPDATE` statement by executing the `DELETE and INSERT` statements.

Example:

```
ERR-CODE: [TDDL-4607][ERR_INSERT_WHEN_UPDATE] Insert new values error,
table is: *****, old Values: *****, new Values: *****
```

After the shard key update feature is enabled, PolarDB-X 1.0 can replace the `UPDATE` statement that updates the shard key with the `DELETE and INSERT` statements. If the execution fails, the TDDL-4607 error is returned.

In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instances. We recommend that you check the status of all backend ApsaraDB RDS for MySQL instances. If the error persists after you resolve the faults in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-4610 ERR_CONNECTION_CLOSED

The error code is returned because a connection is closed.

Example:

```
ERR-CODE: [TDDL-4610][ERR_CONNECTION_CLOSED] connection has been closed
```

After an SQL statement in a transaction fails to be executed or the `KILL` statement is executed to cancel the execution of the SQL statement in the transaction, PolarDB-X 1.0 returns the TDDL-4610 error if you execute other SQL statements over the same database connection.

We recommend that you close the connection that executes the SQL statement and establish a new database connection.

TDDL-1305 ERR_UNKNOWN_SAVEPOINT

The error code is returned because the specified savepoint does not exist.

Example:

```
ERR-CODE: [TDDL-1305][ERR_UNKNOWN_SAVEPOINT] SAVEPOINT ***** does not exist
```

When you execute the `ROLLBACK TO SAVEPOINT` or `RELEASE SAVEPOINT` statement in PolarDB-X 1.0, the TDDL-1305 error is returned if the specified savepoint does not exist.

We recommend that you check whether the savepoint that you specified in the `SAVEPOINT` statement is valid.

TDDL-1094 ERR_UNKNOWN_THREAD_ID

The error code is returned because the session ID that is specified in the `KILL` statement does not exist.

Example:

```
ERR-CODE: [TDDL-1094][ERR_UNKNOWN_THREAD_ID] Unknown thread id: *****
```

When you execute the `KILL` statement in PolarDB-X 1.0 to terminate an SQL statement that is being executed, the TDDL-1094 error is returned if the specified session ID does not exist or the SQL statement is already terminated.

We recommend that you execute the `SHOW PROCESSLIST` statement to query the session ID that corresponds to the SQL statement that you want to terminate and specify the queried session ID in the `KILL` statement.

TDDL-4612 ERR_CHECK_SQL_PRIV

The error code is returned because an SQL statement cannot be executed due to insufficient permissions.

Example:

```
ERR-CODE: [TDDL-4612][ERR_CHECK_SQL_PRIV] check user ***** on db ***** sql
privileges failed.
```

PolarDB-X 1.0 provides a system that allows you to grant permissions to accounts. This system is similar to the account and permission system in MySQL. Only the accounts that are granted the required permissions can be used to execute the SQL statement. If the account that you use is not granted the required permissions, PolarDB-X 1.0 returns the TDDL-4612 error.

We recommend that you check the permissions that the account is granted on the PolarDB-X 1.0 database. If the account is not granted the required permissions, grant the permissions in the PolarDB-X 1.0 console.

TDDL-4613 ERR_INSERT_SELECT

The error code is returned because an error has occurred when PolarDB-X 1.0 executes the `INSERT ... SELECT` statement.

Example:

```
ERR-CODE: [TDDL-4613][ERR_INSERT_SELECT] insert error, table is: *****,
values: *****
```

PolarDB-X 1.0 allows you to split the `INSERT ... SELECT` statement that is executed across databases into the SELECT and INSERT statements and batch execute the statements. If the execution fails, the TDDL-4613 error is returned.

In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instances. We recommend that you check the status of all backend ApsaraDB RDS for MySQL instances. If the error persists after you resolve the faults in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-4614 ERR_EXECUTE_ON_MYSQL

The error code is returned because an error has occurred when PolarDB-X 1.0 executes the SQL statement on a backend ApsaraDB RDS for MySQL instance.

Example:

```
ERR-CODE: [TDDL-4614][ERR_EXECUTE_ON_MYSQL] Error occurs when execute on GROUP '*****': Dup
licate entry '*****' for key 'PRIMARY'
```

This error code is returned if an error occurs when PolarDB-X 1.0 executes an SQL statement on a backend ApsaraDB RDS for MySQL instance. The end part of the returned response contains the error message that is returned from the backend ApsaraDB RDS for MySQL instance. The following messages are sample error messages that are returned from a backend ApsaraDB RDS for MySQL instance:

- `Duplicate entry '*****' for key 'PRIMARY'` indicates that a primary key conflict has occurred when the system writes data to the data table in the ApsaraDB RDS for MySQL instance.
- `The table '*****' is full` indicates that the storage of the temporary table that is used by ApsaraDB RDS for MySQL is full. You must resize the temporary table or optimize the SQL statement.
- `Deadlock found when trying to get lock;` indicates that a dead lock has occurred in the ApsaraDB RDS for MySQL instance. In most cases, dead locks are caused because transaction conflicts occur when the system writes data.

We recommend that you troubleshoot the error based on the error messages that are returned from the ApsaraDB RDS for MySQL instance. For more information about the error messages that are related to SQL statements, see MySQL 5.6 documentation.

If this error persists after you troubleshoot the issues in your application or backend ApsaraDB RDS for MySQL instance, Submit a ticket.

TDDL-4615 ERR_CROSS_JOIN_SIZE_PROTECTION

The error code is returned because the number of rows that are returned for a distributed JOIN query exceeds the upper limit.

Example:

```
ERR-CODE: [TDDL-4615][ERR_CROSS_JOIN_SIZE_PROTECTION] across join table size protection, ch
eck your sql or enlarge the limination size .
```

When PolarDB-X 1.0 runs a distributed JOIN query in nested loops, a large number of memory resources are used if large amounts of data is returned from the right table. This affects the stability of PolarDB-X 1.0 in a negative manner. In PolarDB-X 1.0, the maximum number of rows that can be returned from a right table is 5,000. If this limit is exceeded, PolarDB-X 1.0 returns the TDDL-4615 error.

We recommend that you optimize the SQL statement to prevent large amounts of data from being returned from the right table, or use a better algorithm such as the sort-merge join algorithm to perform distributed JOIN operations in PolarDB-X 1.0.

If you need to change this limit for a specific SQL statement, we recommend that you follow these rules:

- We recommend that you do not change this limit if the size of a single record exceeds 100 KB.
- You can change this limit if the size of a single record is smaller than or equal to 100 KB. We recommend that you do not specify a large value to avoid memory exhaustion.
- If the size of a single record is 100 KB, 500 MB (100 KB × 5,000) of memory resources are required to perform a distributed JOIN query. If this SQL statement is executed over multiple connections, memory resources are prone to be exhausted. For example, if this SQL statement is executed over five connections at the same time, 2.5 GB (500 MB × 5) of memory resources are required.
- To change this limit for an SQL statement, add a hint before the SQL statement. For example, specify `/*!TDDL:MAX_ROW_RETURN_FROM_RIGHT_INDEX_NESTED_LOOP=5100*/SQL` to change the limit to 5,100.
- To globally change this limit, Submit a ticket.

TDDL-4616 ERR_UNKNOWN_DATABASE

The error code is returned because the specified database name is invalid.

Example:

```
ERR-CODE: [TDDL-4616][ERR_UNKNOWN_DATABASE] Unknown database '*****'
```

PolarDB-X 1.0 allows you to specify a database name in a DDL statement. If the database name that you specify is not the same as the database name provided by PolarDB-X 1.0, the TDDL-4616 error is returned.

We recommend that you change the database name in the DDL statement to ensure that the database name that you specify is the same as the database name provided by PolarDB-X 1.0.

TDDL-4617 ERR_SUBQUERY_LIMIT_PROTECTION

The error code is returned because the number of returned rows for a subquery exceeds the upper limit.

Example:

```
ERR-CODE: [TDDL-4617][ERR_SUBQUERY_LIMIT_PROTECTION] The number of rows returned by the sub
query exceeds the maximum number of 20000.
```

When PolarDB-X 1.0 executes an SQL statement that contains a subquery, a large number of memory resources are used if large amounts of data is returned for the subquery. This affects the stability of PolarDB-X 1.0 in a negative manner. In PolarDB-X 1.0, the maximum number of rows that can be returned for a subquery is 20,000. If this limit is exceeded, PolarDB-X 1.0 returns the TDDL-4617 error.

We recommend that you optimize the subquery in the SQL statement to prevent large amounts of data from being returned. You can also rewrite the subquery into a JOIN query so that PolarDB-X 1.0 uses a more suitable algorithm such as the sort-merge join algorithm to perform JOIN operations.

If you need to change this limit, Submit a ticket.

TDDL-4800 ERR_SET_TXCID

The error code is returned because the system has failed to execute the `SET TXC_ID` statement.

Example:

```
ERR-CODE: [TDDL-4800][ERR_SET_TXCID] set txc_id failed: *****
```

TDDL-4801 ERR_TXCID_NULL

This error code is returned because NULL is returned when the `SELECT LAST_TXC_ID` statement is executed.

Example:

```
ERR-CODE: [TDDL-4801][ERR_TXCID_NULL] txc_xid is null: *****
```

TDDL-4802 ERR_SELECT_LAST_TXCID

The error code is returned because the system has failed to execute the `SELECT LAST_TXC_ID` statement.

Example:

```
ERR-CODE: [TDDL-4802][ERR_SELECT_LAST_TXCID] select last_txc_xid failed: *****
```

TDDL-4994 ERR_FLOW_CONTROL

The error code is returned because request throttling is triggered.

Example:

```
ERR-CODE: [TDDL-4994][ERR_FLOW_CONTROL] [*****] flow control by *****
```

This error code indicates that the number of SQL requests processed by PolarDB-X 1.0 has reached the upper limit and the current request is rejected.

We recommend that you check whether the peak value of the number of SQL requests is as expected. If this error persists when the number of SQL requests decreases to be lower than the upper limit, Submit a ticket.

TDDL-4998 ERR_NOT_SUPPORT

The error code is returned because the feature is not supported.

Example:

```
ERR-CODE: [TDDL-4998][ERR_NOT_SUPPORT] ***** not support yet!
```

This error code indicates that the SQL syntax or the feature that you use is not supported by PolarDB-X 1.0.

If the SQL syntax or the feature is required by your business, Submit a ticket.

TDDL-5001 ERR_TRANS

The error code is returned because a common transaction error has occurred.

Example:

```
ERR-CODE: [TDDL-5001][ERR_TRANS] Too many lines updated in statement.
```

Resolve the error based on the error message. `Too many lines updated in statement` indicates that the number of rows that you want to update by executing the UPDATE statement exceeds the upper limit of 1,000. We recommend that you check the `WHERE` clause in the UPDATE statement. If you need to update a large amount of data in a transaction, you can use the `/*TDDL:UNDO_LOG_LIMIT={number}*/` hint that is provided by PolarDB-X 1.0 to change the upper limit.

`Deferred execution is only supported in Flexible or XA Transaction` indicates that the deferred execution feature is available only for flexible or XA transactions. Before you use `/*TDDL:DEFER*/` to enable deferred execution, execute the `SET drds_transaction_policy = ***` statement to change the transaction policy of your PolarDB-X 1.0 instance.

For information about other error messages, see Submit a ticket.

TDDL-5002 ERR_TRANS_UNSUPPORTED

The error code is returned because the syntax or the feature used in the transaction is not supported.

Example:

```
ERR-CODE: [TDDL-5002][ERR_TRANS_UNSUPPORTED] Table without primary keys is not
supported.
```

This error code indicates that this feature is not supported for PolarDB-X 1.0 transactions. If you need to use this feature, Submit a ticket.

TDDL-5003 ERR_TRANS_LOG

The error code is returned because transaction logs cannot be accessed.

Example:

```
ERR-CODE: [TDDL-5003][ERR_TRANS_LOG] Failed to update transaction state: *****
```

When PolarDB-X 1.0 performs a distributed transaction, PolarDB-X 1.0 accesses the transaction logs in the backend ApsaraDB RDS for MySQL instances. This helps ensure the atomicity of the distributed transaction. If PolarDB-X 1.0 fails to read or write transaction logs, the TDDL-5003 error is returned.

In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instances. We recommend that you check the status of and access workloads on the backend ApsaraDB RDS for MySQL instances of your PolarDB-X 1.0 instance. If this error persists after you resolve the fault in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-5004 ERR_TRANS_NOT_FOUND

The error code is returned because the specified transaction ID does not exist.

Example:

```
ERR-CODE: [TDDL-5008][ERR_TRANS_TERMINATED] Current transaction was killed
or timeout. You may need to set a longer timeout value.
```

This error code indicates that the specified transaction is terminated by the KILL statement or the execution has timed out. The timeout period of a transaction is specified by the drds_transaction_timeout parameter.

If this error is returned due to a transaction timeout, we recommend that you execute the SET drds_transaction_timeout = *** statement to change the timeout period for the transaction. Specify the value of the drds_transaction_timeout parameter in milliseconds.

TDDL-5006 ERR_TRANS_COMMIT

The error code is returned because PolarDB-X 1.0 has failed to commit a transaction.

Example:

```
ERR-CODE: [TDDL-5006][ERR_TRANS_COMMIT] Failed to commit primary group *****:
*****, TRANS_ID = *****
```

If an error occurs when PolarDB-X 1.0 commits a transaction, the transaction is automatically rolled back. TRANS_ID indicates the ID of the transaction.

In most cases, this error occurs because a fault occurs in the backend ApsaraDB RDS for MySQL instances. We recommend that you check the status of and access workloads on the backend ApsaraDB RDS for MySQL instances of your PolarDB-X 1.0 instance. If this error persists after you resolve the fault in the backend ApsaraDB RDS for MySQL instances, Submit a ticket.

TDDL-5007 ERR_TRANS_PARAM

The error code is returned because the specified transaction parameter is invalid.

Example:

```
ERR-CODE: [TDDL-5007][ERR_TRANS_PARAM] Illegal timeout value: *****
```

This error code indicates that you specified an invalid value for the transaction parameter in the statement. For example, in the `SET drds_transaction_timeout = ***` statement, the drds_transaction_timeout parameter is set to a negative number.

TDDL-5008 ERR_TRANS_TERMINATED

The error code is returned because a transaction is terminated by the KILL statement or due to a timeout.

Example:

```
ERR-CODE: [TDDL-5008][ERR_TRANS_TERMINATED] Current transaction was killed
or timeout. You may need to set a longer timeout value.
```

This error code indicates that the specified transaction is terminated by the KILL statement or the execution has timed out. The timeout period of a transaction is specified by the `drds_transaction_timeout` parameter.

If this error is returned due to a transaction timeout, we recommend that you execute the `SET drds_transaction_timeout = ***` statement to change the timeout period for the transaction. Specify the value of the drds_transaction_timeout parameter in milliseconds.